



# PARAM Yukti

## User's Manual

---

Ver. 1.0

Last updated: July 20, 2021

[www.cdac.in](http://www.cdac.in)

## © Copyright Notice

Copyright © 2021 Centre for Development of Advanced Computing All Rights Reserved.

Any technical documentation that is made available by C-DAC (Centre for Development of Advanced Computing) is the copyrighted work of C-DAC and is owned by C-DAC. This technical documentation is being delivered to you as is, and C-DAC makes no warranty as to its accuracy or use. Any use of the technical documentation or the information contained therein is at the risk of the user. C-DAC reserves the right to make changes without prior notice.

No part of this publication may be copied without the express written permission of C-DAC.

## ® Trademarks

CDAC, CDAC logo, NSM logo are trademarks or registered trademarks.

Other brands and product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.



## Intended Audience

This document is meant for PARAM Yukti users.

## Typographic Conventions

Symbol	Meaning
<u>Blue underlined text</u>	A hyperlink or link you can click to go to a related section in this document or to a URL in your web browser.
<b>Bold</b>	The names of menus, menu items, headings, and buttons.
<i>Italics</i>	Variables or placeholders or special terms in the document.
<code>Console text</code>	Console commands



## Getting help

For technical assistance or license renewal, please send an email to [yuktisupport@jncasr.ac.in](mailto:yuktisupport@jncasr.ac.in).

## Give us your feedback

We value your feedback. Kindly send your comments on the content of this document to [yuktisupport@jncasr.ac.in](mailto:yuktisupport@jncasr.ac.in) . Please include the page number of the document along with your feedback.



## DISCLAIMER

The information contained in this document is subject to change without notice. C-DAC shall not be liable for errors contained herein or for incidental or consequential damages in connection with the performance or use of this manual.

# Contents

<b>Introduction</b>	<b>7</b>
<b>System Architecture and Configuration</b>	<b>8</b>
System Hardware Specifications	8
Login Nodes	8
CPU Compute Nodes	9
GPU Compute Nodes	10
Storage	10
Operating System	10
Primary Interconnection Network	12
Secondary Interconnection Network	12
Software Stack	12
<b>First Things First</b>	<b>15</b>
Getting an Account on PARAM Yukti	15
First login	15
Forgot Password?	16
System Access	16
Remote Access	17
Transferring files between local machine and HPC cluster	18
Tools	20
<b>Running Interactive Jobs</b>	<b>23</b>
<b>Managing Jobs through its Lifecycle</b>	<b>24</b>
walltime	24
List Partition	25
<b>Addressing Basic Security Concerns</b>	<b>31</b>
<b>More about Batch Jobs (SLURM)</b>	<b>32</b>
Parameters used in SLURM job script	32
I am familiar with PBS/ TORQUE. How do I migrate to SLURM?	35
<b>Preparing Your Own Executable</b>	<b>37</b>

<b>Job Scheduling on PARAM Yukti</b>	<b>41</b>
Scheduler	41
sinfo	41
walltime	41
<b>Debugging Your Codes</b>	<b>46</b>
Introduction	46
Basics How-Tos	46
Conclusions	65
Points to Note	65
Overall Coding Modifications Done	66
<b>Machine Learning / Deep Learning Application Development</b>	<b>67</b>
How to Install your own Software?	68
<b>Some Important Facts</b>	<b>70</b>
About File Size	70
Little-Endian and Big-Endian issues?	71
<b>Best Practices for HPC</b>	<b>72</b>
<b>Installed Applications/Libraries</b>	<b>73</b>
Standard Application Programs on PARAM Yukti	73
LAMMPS Applications	74
GROMACS APPLICATION	76
<b>Acknowledging the National Supercomputing Mission in Publications</b>	<b>79</b>
<b>Getting Help – PARAM Yukti Support</b>	<b>80</b>
Steps to Create a New Ticket	80
Closing Your Account on PARAM Yukti	83
<b>References</b>	<b>89</b>

## List of Figures

Figure 1 - PARAM Yukti Architecture Diagram	11
Figure 2 – Software Stack	13
Figure 3 - A snapshot of command using MobaXterm	20
Figure 4 - A snapshot of the “scp” command using Windows command prompt	20
Figure 5 - A snapshot of "scp" command using Windows PowerShell	21
Figure 6 - A snapshot of "scp" tool to transfer file to and from remote computer	22
Figure 7 – Enter Captcha/String	22
Figure 8 - Output of sinfo command	25
Figure 9 – Snapshot depicting the usage of “Job Array”	27
Figure 10 – scontrol show node displays compute node information	28
Figure 11 – scontrol show partition displays specific partition details	28
Figure 12 – scontrol show job displays specific job information	29
Figure 13 – sinfo Command	41
Figure 14 - Listing the shares of association to a cluster	44
Figure 15 – Snapshot of debugging process	49
Figure 16 – Snapshot of debugging process	50
Figure 17- Output at a debugging stage	51
Figure 18 – Snapshot of debugging process	51
Figure 19 – Output depicting “Arithmetic Exception”	52
Figure 20 – Snapshot of debugging process	52
Figure 21 – Well, we dumped core !!	52
Figure 22 - Snapshot of debugging process	53
Figure 23 – Setting Breakpoint	54
Figure 24 – single-stepping through to catch error !!	55
Figure 25 – Debugging continued	56
Figure 26 – Debugging continued	56
Figure 27 – Setting a watchpoint	57
Figure 28 – Debugging continued	58
Figure 29 – Well, Back to square one !!	59
Figure 30 – Again Dumping Core!! Things are getting interesting or frustrating or both !!	59
Figure 31 – Debugging continued	60
Figure 32 – Debugging continued	60
Figure 33 – Debugging continued (Will it ever end?)	61
Figure 34 – We are almost there !!	61
Figure 35 – Debugging continued	62
Figure 36 – At last a clue!!!	63
Figure 37 - Correction applied !!	64
Figure 38 – Resolved !!!	65

Figure 39 – What all we did to get things right!	66
Figure 40 – Snapshot of Ticketing System	80
Figure 41- Snapshot of Ticketing System	81
Figure 42 - Snapshot of Ticketing System	81
Figure 43 - Snapshot of Ticketing System	82

# Introduction

---

This document is the user manual for the PARAM Yukti Supercomputing facility at JNCASR Bangalore. It covers a wide range of topics ranging from a detailed description of the hardware infrastructure to the information required to utilize the supercomputer, such as information about logging on to the supercomputer, submitting jobs, retrieving the results on to user's Laptop/ Desktop, etc. In short, the manual describes all that one needs to know to effectively utilize PARAM Yukti.

The supercomputer PARAM Yukti is based on the heterogeneous and hybrid configuration of Intel Xeon Cascade lake processors and NVIDIA Tesla V100. The system was designed and implemented by the HPC Technologies team, Centre for Development of Advanced Computing (C-DAC).

It consists of 2 Master nodes, 4 Login nodes, 4 Service nodes and 156 (CPU+GPU) compute nodes with a total peak computing capacity of **838** (CPU+GPU) **TFLOPS** performance.



# System Architecture and Configuration

## System Hardware Specifications

PARAM Yukti system is based on processor Intel Xeon Platinum 8268 and Intel Xeon Gold 6248 and with a total peak performance of 838 TFLOPS. The cluster consists of compute nodes connected with BullSequana XH2000 HDR 100 Infiniband interconnect network. The system uses the Lustre parallel file system.

- Total number of nodes: 166 (10 + 156)
  - Master nodes: 2
  - Login nodes: 4
  - Service nodes: 4
  - CPU only nodes: 75
  - GPU ready nodes: 32
  - GPU nodes: 10
  - High Memory nodes:39

## Login Nodes

Login nodes are typically used for administrative tasks such as editing, writing scripts, transferring files, managing your jobs etc. You will always get connected to one of the login nodes. From the login nodes, you can connect to a compute node and execute an interactive job or submit the batch jobs through the batch system (SLURM) to run your jobs on compute nodes. For ALL users PARAM Yukti login nodes are the entry points and hence are shared. By default, there will be a limit on the CPU time that can be used on a login node by a user and there is a limit/user on the memory as well. If any of these are exceeded, the job will get terminated.

### Login Nodes: 4

**2\* Intel Xeon G-6248**  
 Cores = 40, 2.5 GHz  
 Memory= 384 GB  
 HDD = 1 TBx8

Total Cores = 160 cores

Total Memory = 1536 GB

## CPU Compute Nodes

CPU nodes are indeed the workhorses of PARAM Yukti. All the CPU-intensive activities are carried on these nodes. Users can access these nodes from the login node to run interactive or batch jobs. Some of the nodes have higher memory, which can be exploited by users in an aforementioned way.

#### CPU only Compute Nodes:

75

**2\* Intel Xeon Platinum 8268**                      Total Cores = 3600 cores  
 Cores = 48, 2.9 GHz  
 Memory= 192 GB, DDR4 2933 MHz              Total Memory=14400 GB  
 SSD = 480 GB (local scratch) per  
 node

---

#### GPU ready Compute

Nodes: 32

**2\* Intel Xeon Platinum 8268**                      Total Cores = 1536 cores  
 Cores = 48, 2.9 GHz  
 Memory= 192 GB, DDR4 2933 MHz              Total Memory=6144 GB  
 SSD = 480 GB (local scratch) per node

---

## GPU Compute Nodes

GPU compute nodes are the nodes that have CPU cores along with accelerators cards. For some applications, GPUs get markedly high performance. For exploiting these, one has to make use of special libraries which map computations on the Graphical Processing Units (Typically one has to make use of CUDA or OpenCL).

#### GPU Compute Nodes:

10

**2\* Intel Xeon G-6248**                      Total Cores = 400 cores  
 Cores = 40, 2.5 GHz  
 Memory= 192 GB, DDR4 2933 MHz              Total Memory= 1920 GB  
  
 SSD = 480 GB per node  
**2\*nVidia V100 per node**  
 GPU Cores per node= 2\*5120=  
 10240  
 GPU Memory = 16 GB HBM2 per nVidia V100

---

#### CPU only Compute Nodes with High memory : 39

**2\* Intel Xeon Platinum 8268**  
 Cores = 48, 2.9 GHz                      Total Cores = 1872 cores  
 Memory= 768 GB, DDR4 2933 MHz              Total Memory=29952 GB

SSD = 480 GB per node

## Storage

- Based on Lustre parallel file system
- The total usable capacity of 1200 TB Primary storage
- Throughput 25GB/s

## Operating System

- The operating system on PARAM Yukti is Linux – CentOS 7.6

## PARAM Yukti Architecture Diagram

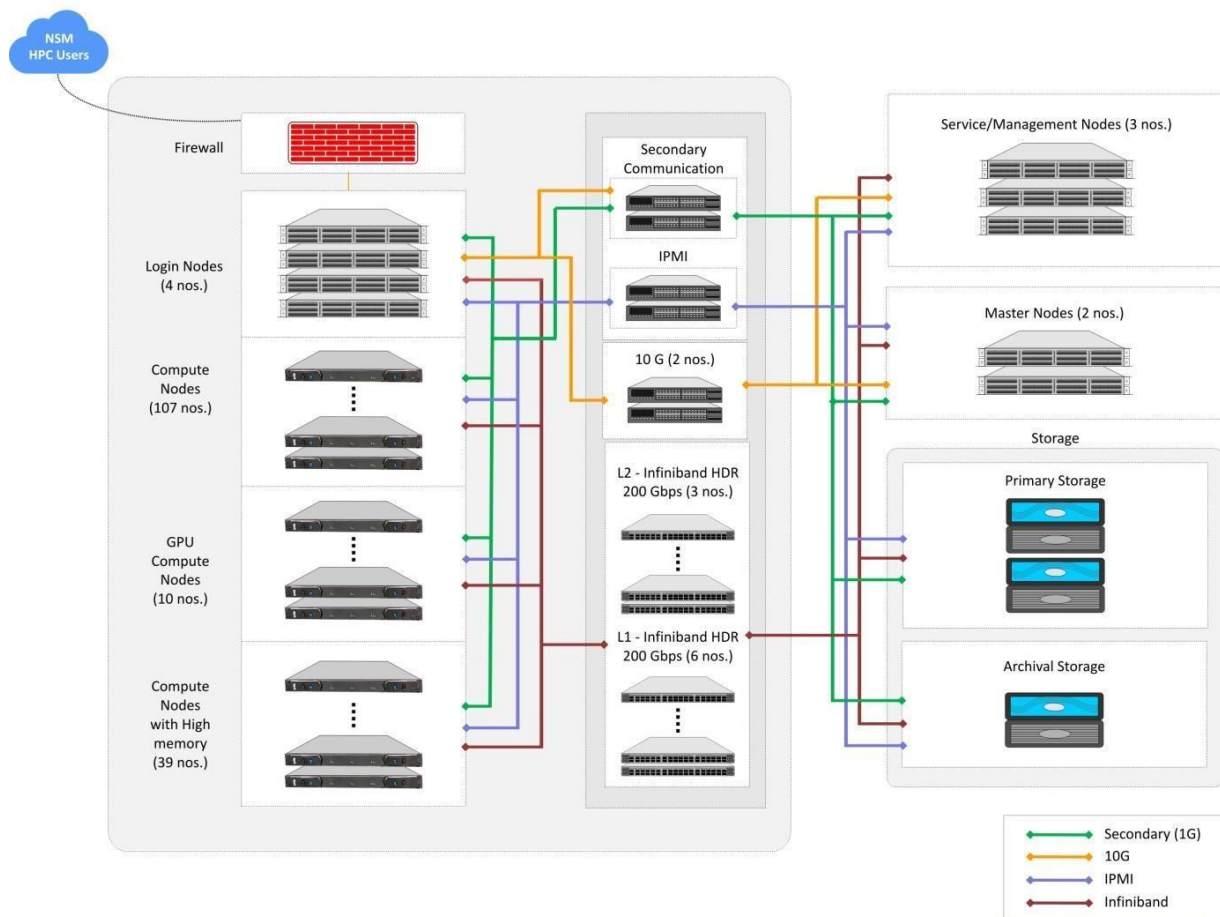


Figure 1 - PARAM Yukti Architecture Diagram

## Network infrastructure

Robust network infrastructure is essential to implement the basic functionalities of a cluster. These functionalities are:

- Management functionalities i.e. to monitor, troubleshoot, start, stop various components of the cluster, etc. (Network/ portion of Network which

implements this functionality is referred to as Management fabric).

- b) Ensuring fast read/ write access to the storage (Network/ portion of Network which implements this functionality is referred to as storage

fabric).

- c) Ensuring fast I/O operations like connecting to other clusters, connecting the cluster to various users on the campus LAN, etc. (Network/ portion of Network which implements this functionality is referred to as I/O Fabric).
- d) Ensuring High-Bandwidth, Low-latency communication amongst processors for achieving high-scalability (Network/ portion of Network which implements this functionality is referred to as Message Passing Fabric)

Technically, ALL the aforementioned functionalities can be implemented in a single

network. From the perspectives of requirements, optimal performance, and economic suitability, the aforementioned functionalities are implemented using two different networks based on different technologies, as mentioned next:

## Primary Interconnection Network

Computing nodes of PARAM Yukti are interconnected by a high-bandwidth, low-latency interconnect network.

### InfiniBand: HDR 100 Gbps

InfiniBand is a high-performance communication architecture owned by Mellanox. This communication architecture offers low communication latency, low power consumption, and high throughput. All CPU nodes are connected via the InfiniBand interconnect network.

## Secondary Interconnection Network

### Gigabit Ethernet: 1 Gbps

Gigabit Ethernet is the interconnection network that is most commonly available. For Gigabit Ethernet, no additional modules or libraries are required. The Open MPI, MPICH implementations will work over Gigabit Ethernet.

## Software Stack

**A software stack** is an aggregation of software components that work in tandem to accomplish a given task. The task can be, to facilitate a user to execute his job/s or to facilitate a system administrator to manage a system efficiently. In effect, the software will have all the necessary components to accomplish a given task. There

may be multiple components of different flavors to accomplish a given sub-task. The user/ administrator may mix and match these components depending on his choice. Typically, a user would be interested in preparing his executable, executing the same with his data sets, and visualize the output generated by him. For accomplishing the same, the user would need to compile his codes, link the codes with communication

libraries, Math Libraries, Numerical algorithm libraries, prepare the executables, run the same with desired data sets, monitor the progress of his jobs, gathering the results, and visualizing the output.

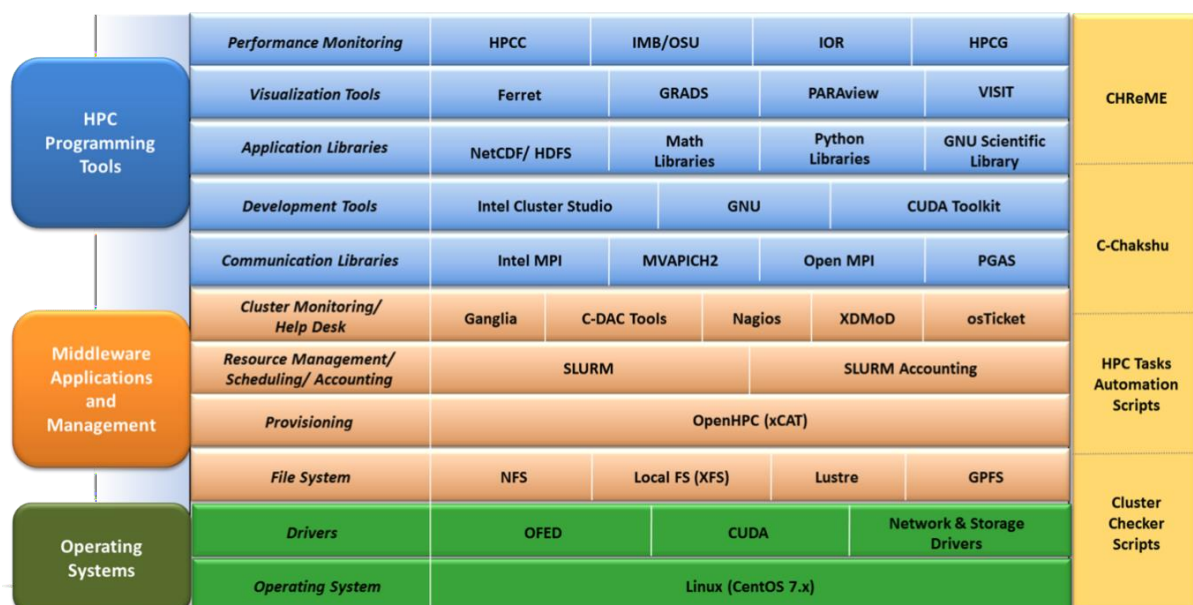
Typically, a system administrator would be interested in ensuring that all the resources are optimally utilized. For accomplishing this, he may need some installation tools, tools for checking the health of all the components, good schedulers, tools to facilitate allocation of resources to users, and monitor the usage of the resources.

The software stack provided with this system has a gamut of software components that meets all the requirements of a user and that of a system administrator. The components of the software stack are depicted in figure 2.

Amongst these, C-CHAKSHU and CHReME have been recently developed and deployed by CDAC. We solicit your feedback on these tools at [yuktisupport@jncasr.ac.in](mailto:yuktisupport@jncasr.ac.in)

**C-C HAKSHU:** This is a multi-cluster management tool that facilitates the administrator to efficiently operate the HPC facility. It also enables the user to monitor system metrics relating to CPU, Storage, Interconnects, File system and Application specific utilization from a single dashboard. For more information, please follow the link. <http://paramyukti.jncasr.ac.in:4200>

**CHReME:** This is a web-based resource management portal with an intuitive GUI that facilitates managing, monitoring, and tracking HPC workloads. It provides a customizable and user-friendly workflow interface to HPC users for state-based pre-processing, execution, and post-processing of applications. For more details, please follow the link. <http://paramyukti.jncasr.ac.in:8097/CHReME>



<b>Functional Areas</b>	<b>Components</b>
Base OS	CentOS 7.6
Architecture	X86_64
Provisioning Cluster Manager	xCAT 2.14.6 Openhpc (ohpc-xCAT 1.3.8)
Monitoring Tools	C-CHAKSHU, Nagios, Ganglia, XDMoD
Resource Manager	Slurm
I/O Services	Lustre Client
High Speed Interconnects	Mellanox InfiniBand
Compiler Families	GNU (gcc, g++, gfortran) Intel Compiler (icc, ifort, icpc)
MPI Families	MVAPICH, OpenMPI, MPICH

---

# First Things First

---

## Getting an Account on PARAM Yukti

To begin with, you need to get an account on PARAM Yukti. This is a very easy process. Please follow the steps given below:

- a) Download the 'User Account Creation Form' by following the link <https://paramyukti.jncasr.ac.in/ucform>
- b) Fill in the relevant details.
- c) Get the signatures of your Head of the Department and the 'Approving Authority'.

Note:

- For JNCASR users will have the approving authority from JNCASR Bangalore. They can submit it to the PARAM Yukti system Administrator\* or email a scanned copy to [yuktisupport@jncasr.ac.in](mailto:yuktisupport@jncasr.ac.in)
  - For Users who are not from JNCASR, Bangalore will have to email the scanned copy to [yuktisupport@jncasr.ac.in](mailto:yuktisupport@jncasr.ac.in), HoD HPC-Tech CDAC will be the approving authority for them.
- d) You will receive an Email in your official Email ID intimating the creation of your account along with a temporary password set by the system to your account. You will also get a copy of this document by Email.
  - e) Log into PARAM Yukti and you will be prompted to change the password. When once you change the temporary password provide by the system to your own password, you are ready to use PARAM Yukti!!

Info: \*

Param Yukti, Jawaharlal Nehru Centre For Advanced Scientific Research (JNCASR) Rachenahalli Lake Rd, Jakkur, Bengaluru, Karnataka 560064 Email: <a href="mailto:yuktisupport@jncasr.ac.in">yuktisupport@jncasr.ac.in</a>
--

## First login

Whenever the newly created user on PARAM Yukti tries to login with the User Id and password (temporary, system-generated) provided over the Email through PARAM Yukti support, he/she will next be prompted to create a "new password" of their choice which will change the temporary, system-generated password. This will enable you to keep your account secure. It is recommended that you have a strong

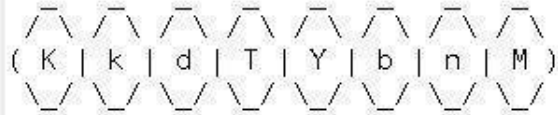
password that contains the



combination of alphabets (lower case / upper case), numbers, and a few special characters that you can easily remember.

Given next is a screenshot that describes the scenario for “the first login”

Observe the picture below and answer the question listed afterwards:



Type the string above: KkdTYbnM

Password:

You are required to change your password immediately (password aged)  
password expired 18078 days ago

New password:

Your password will be valid for 90 days. On the expiry of 90 days period, you will be prompted to change your password, on attempting to log in. You are required to provide a new password.

## Forgot Password?

There is nothing to panic!! Please raise a ticket regarding this issue and the system administrators will resolve your problem. Please refer to the section “Getting Help – PARAM Yukti Support, described elsewhere in this manual. Follow the GUI-based, user-friendly ticketing system. Please follow the steps given below:

1. Open the PARAM Yukti support site i.e the ticketing tool by following the link <https://paramyukti.jncasr.ac.in/support>
2. Login with your registered email id, Complete name, Contact number.
3. There you can raise a ticket to get the password reset.
4. The system admin person will revert with an email for verification.
5. Once acknowledged, the password is reset for the user and an email is sent back for intimating the same.
6. Then the user can login with the temporary password and can set a new password of his/her choice.

## System Access

### Accessing the cluster

The cluster can be accessed through 4 general login nodes, which allows users to login.

- You may access the login node through ssh.

- The login node is the primary gateway to the rest of the cluster, which has a job scheduler (called Slurm). You may submit jobs to the queue and they will run when the required resources are available.
- Please do not run programs directly on the login node. Login node is used to submit jobs, transfer data, and compile source code. (If your compilation takes more than a few minutes, you should submit the compilation job into the queue to be run on the cluster.)
- By default, two directories are available (i.e. /home and /scratch). These directories are available on login node as well as the other nodes on the cluster. /scratch is for temporary data storage, generally used to store data required for running jobs.

## Remote Access

### Using SSH in Windows

To access PARAM Yukti you need to “ssh” the login server. PuTTY is the most popular open-source “ssh” client application for Windows, you can download it from (<http://www.putty.org/>). Once installed, find the PuTTY application shortcut in your Start Menu, desktop. On clicking the PuTTY icon The PuTTY Configuration dialog should appear. Locate the “Host Name” input Field in the PuTTY Configuration screen. Enter the user name along with the IP address or Hostname with which you wish to connect.

(e.g. [username]@[Host Name] -p [port number]) for outside

access (e.g. [username]@[Host Name]) for local access

Enter your password when prompted, and press Enter.

### Using SSH in Mac or Linux

Both Mac and Linux systems provide a built-in SSH client, so there is no need to install any additional package. Open the terminal, connect to an SSH server by typing the following command:

```
ssh [username]@[hostname]
```

For example, to connect to the PARAM Yukti Login Node, with the username

```
user1: ssh user1@ [redacted] -p [redacted]
```

You will be prompted for a password, and then will be connected to the server.

## Password

How to change the user password?

Use the **passwd** command to change the password for the user from login node.

```
[root@login1 ~]# passwd
Changing password for user
(current) LDAP Password:
New password:
Retype new password:
```

## Transferring files between local machine and HPC cluster

Users need to have the data and application related to their project/research work on PARAM Yukti.

To store the data special directories have been made available to the users with the name “home” the path to this directory is “/home”. Whereas these directories are common to all the users, a user will get his own directory with their username in /home/ directories where they can store their data.

```
/home/<username>/: ! This directory is generally used by the user
to install applications.
```

However, there is a limit to the storage provided to the users, the limits have been defined according to quota over these directories, all users will be allotted the same quota by default. When a user wishes to transfer data from their local system (laptop/desktop) to the HPC system, they can use various methods and tools.

A user using ‘Windows’ operating system will get methods and tools that are native to Microsoft windows and tools that could be installed on your Microsoft Windows machine. Linux operating system users do not require any tool. They can just use the “scp” command on their terminal, as mentioned below.

Users are advised to keep a copy of their data with themselves, once the project/research work is completed by transferring the data in from PARAM Yukti to their local system (laptop/desktop). The command shown below can be used for effecting file transfers (In all the tools):

```
scp -r <path to the local data directory> <your username>@<IP of
paramyukti/Host Name>:<path to directory on HPC where to save
the data>
```

Example:

The same command could be used to transfer data from the HPC system to your local system (laptop/desktop).

```
scp -r /dir/dir/file testuser@<cluster IP/Name>:/home/testuser
```

Example:

```
scp -r <path to directory on HPC> <your username>@<IP of local system>:<path to the local data directory>
```

```
scp -r /home/testuser testuser@<local system IP/Name>:/dir/dir/file
```

**Note:** The Local system (laptop/desktop) should be connected to the network with which it can access the HPC system.

To reiterate,

Copying Directory/File from local machine to PARAM Yukti:

To copy a local directory from your Linux system (say Wrf-2.0) to your home directory in your PARAM Yukti HPC account, the procedure is:

1. From the terminal go to the parent directory using `cd` command. `user1@mylaptop:~$cd ~/MyData/`
2. Under parent directory type `ls` & press Enter key, & notice Wrf-2.0 is there. `user1@mylaptop: ~$ls Files TempFiles-0.5 Wrf-2.0`
3. Begin copy by typing:  
`user1@mylaptop:~$ scp -r Wrf-2.0 (username)@paramyukti.jncasr.ac.in`  
< you will be prompted for password ; enter your password >
4. Now login to your account as: `user1@mylaptop:~$ ssh (your username)@ paramyukti.jncasr.ac.in` < you will be prompted for password ; enter password > `[user1@login ~]$`
5. `ls` command, you should see Wrf-2.0 directory.
6. While copying from PARAM Yukti to your local machine, follow the same steps

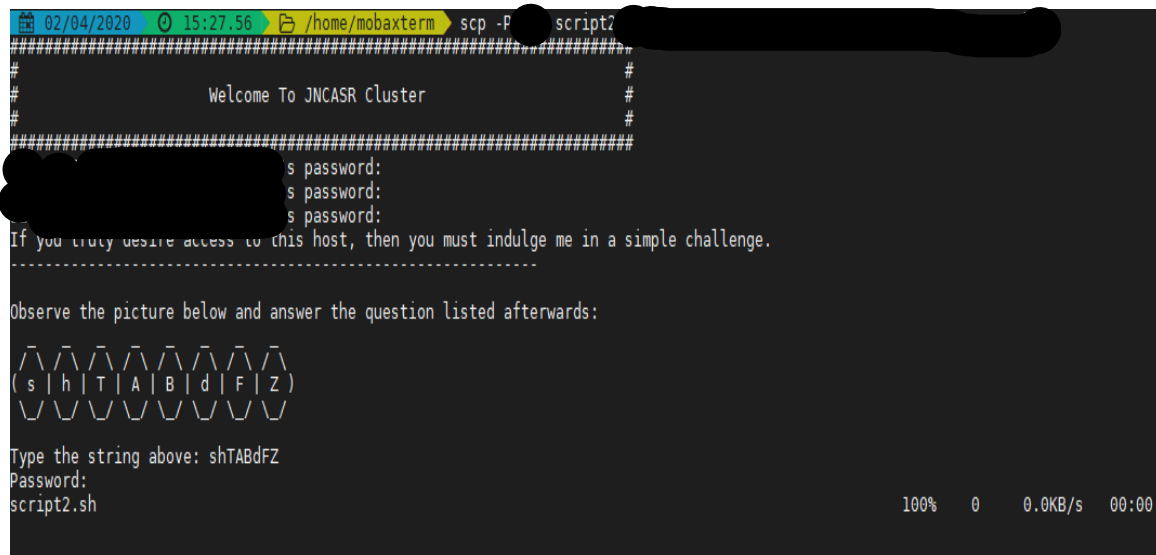
By interchanging source and destination in the scp command. Refer to the generic copying described earlier.

## Tools

### MobaXterm (Windows installable application):

It is a third-party freely available tool that can be used to access the HPC system and transfer files to the PARAM Yukti system through your local systems (laptop/desktop).

Link to download this tool: <https://mobaxterm.mobatek.net/download-home-edition.html>



```

02/04/2020 15:27:56 /home/mobaxterm scp -P script2
#####
#
# Welcome To JNCASR Cluster
#
#
#####
s password:
s password:
s password:
If you truly desire access to this host, then you must indulge me in a simple challenge.
-----
Observe the picture below and answer the question listed afterwards:

  ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
 ( s | h | T | A | B | d | F | Z )
  ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

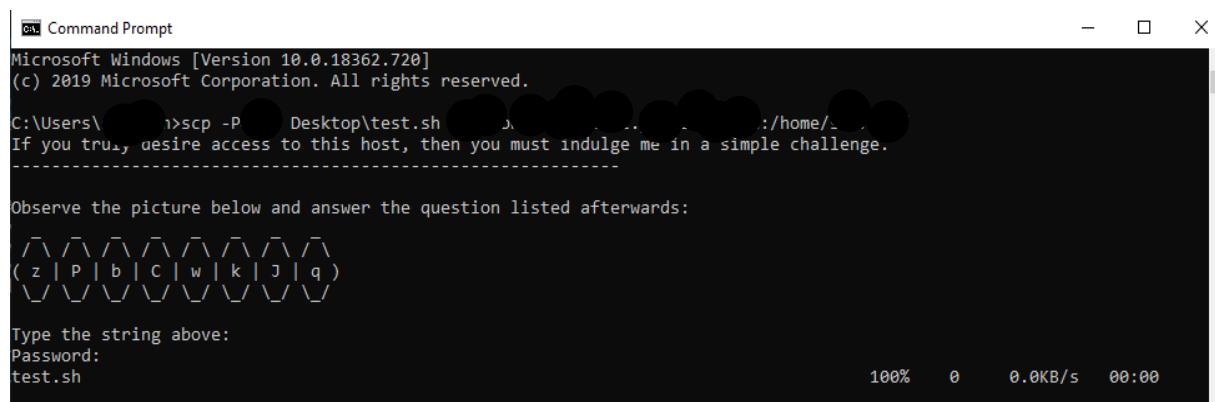
Type the string above: shTABdFZ
Password:
script2.sh 100% 0 0.0KB/s 00:00

```

Figure 3 - A snapshot of command using MobaXterm

### Command Prompt (Windows native application):

This is a native tool for Windows machines that can be used to transfer data from the PARAM Yukti system through your local systems (laptop/desktop).



```

Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\>scp -P Desktop\test.sh :/home/
If you truly desire access to this host, then you must indulge me in a simple challenge.
-----
Observe the picture below and answer the question listed afterwards:

  ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
 ( z | P | b | C | w | k | J | q )
  ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

Type the string above:
Password:
test.sh 100% 0 0.0KB/s 00:00

```

Figure 4 - A snapshot of "scp" command using Windows command prompt.



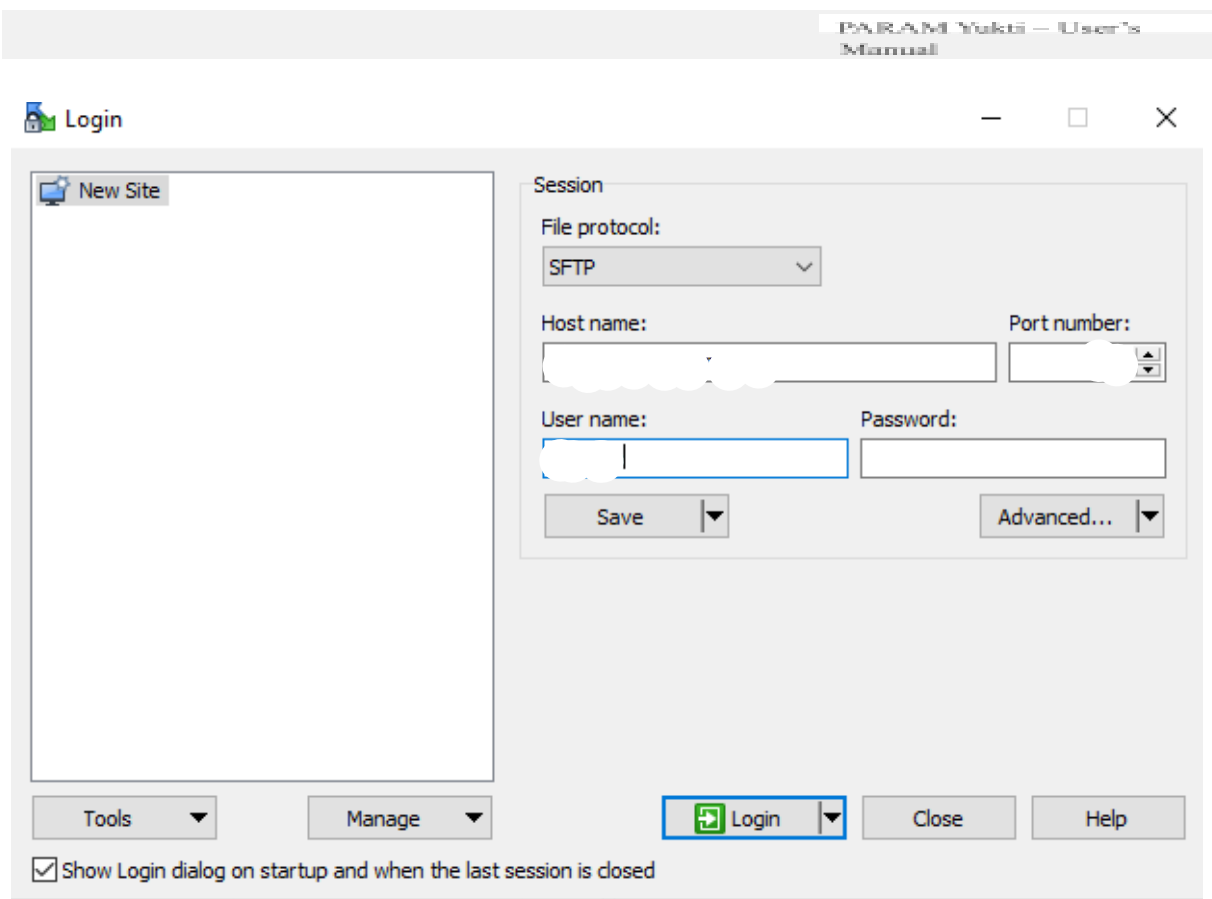


Figure 6 - A snapshot of "scp" tool to transfer files to and from a remote computers.

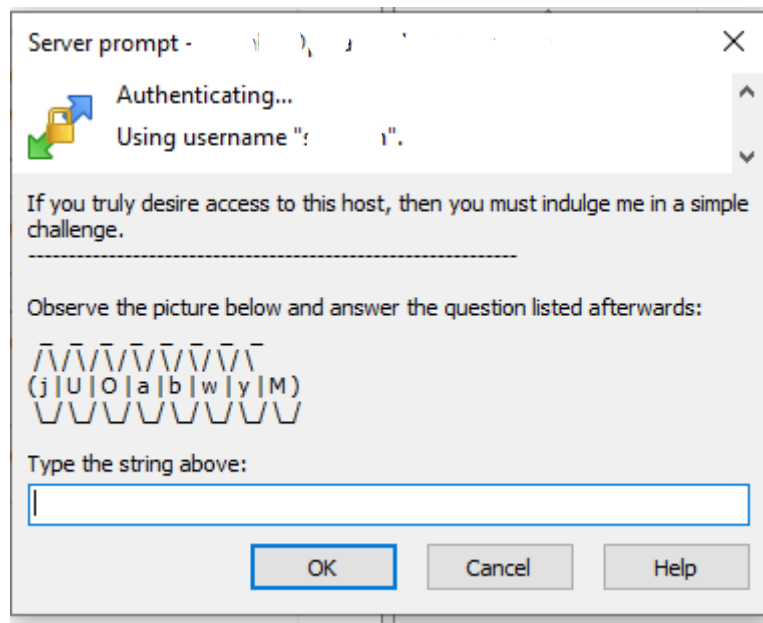


Figure 7 – Enter Captcha/String

**Note:** Port Used for SFTP connection is          and not 22. Please change it to

# Running Interactive Jobs

In general, the jobs can be run in an interactive manner or in batch mode. You can run an interactive job as follows:

The following command asks for a single core on one hour with the default amount of memory.

```
$ srun --nodes=1 --ntasks-per-node=1 --time=01:00:00 --pty bash -i
```

The command prompt will appear as soon as the job starts. This is how it looks once the interactive job starts:

```
srun: job xxxxx queued and waiting for resources srun: job xxxxx  
has been allocated resources
```

Where xxxxx is the job id.

Exit the bash shell to end the job. If you exceed the time or memory limits the job will also abort.

Please note that PARAM Yukti is NOT meant for executing interactive jobs. However, for the purpose of quickly ascertaining the successful run of a job before submitting a large job in batch (with large iteration counts), this can be used. This can even be used for running small jobs. The point to be kept in mind is that, since others too would be using this node, it is prudent not to inconvenience them by running large jobs.

It is a good idea to specify the CPU account name as well (if you face any problems)

```
$ srun --account=<NAME_OF_MY_ACCOUNT> --nodes=1 --ntasks-per-node=1  
--time=01:00:00 -- pty bash -i
```



# Managing Jobs through its Lifecycle

PARAM Yukti extensively uses modules. The purpose of the module is to provide the production environment for a given application, outside of the application itself. This also specifies which version of the application is available for a given session. All applications and libraries are made available through module files. A user has to load the appropriate module from the available modules. Users can add a particular module in their `~/.bashrc` also if they don't want to load a particular module file for each time after they login.

```

module                                # This command lists all the available
..                                     odul
module load                            # This will load the intel compilers into
compiler/intel/2018.2.199              your
..
module unloadcompiler/intel/2018.2.199 # This will remove all environment setting
related to intel-2018.compiler loaded previously

```

A simple Slurm job script

```

#!/bin/sh
#SBATCH -N                               // specifies number of
#SBATCH --ntasks-per-node=48             // specifies cores per node
#SBATCH --time=06:50:20 // specifies maximum duration of run
#SBATCH --job-name=lammps                // specifies job name
#SBATCH --error=job.%J.err_node_48      // specifies error file
name #SBATCH --output=job.%J.out_node_48 //specifies output
file name #SBATCH --partition=standard   // specifies
..
cd                                         // To run job in the directory from where it
..
export I_MPI_FABRICS=shm:dapl //For Intel MPI versions 2019 onwards
this value must be shm:ofi
mpirun.hydra -n $SLURM_NTASKS lammps.exe

```

## walltime

Walltime parameter defines how long your job will run. The maximum runtime of a job is allowed as per QoS policy. If more than 3 days are required, a special request needs to be sent to the HPC coordinator and it will be dealt with on a case-to-case basis. The command line to specify walltime is given below.

```
srun -t walltime <days-hours:mins:seconds>
```

and also as part of the submit scripts described in the manual. If a job does not get completed within the walltime specified in the script, it will get terminated.

The biggest advantage of specifying appropriate walltime is that the efficiency of scheduling improves resulting in improved throughput in all jobs including yours. You are encouraged to arrive at the appropriate walltime for your job by executing your jobs few times.

---

**NOTE:** You are requested to explicitly specify the walltime in your command lines and scripts.

---

## List Partition

sinfo displays information about nodes and partitions(queues).

**\$ sinfo**

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
standard* up 3-00:00:00    3  maint cn[108-110]
standard* up 3-00:00:00   21  mix  cn[080-089],gpu[001-010],hm034
standard* up 3-00:00:00  131  alloc cn[001-043,045-079,090-098,100-101,103-107],hm[001-033,036-039]
standard* up 3-00:00:00    4  idle  cn[044,099,102],hm035
gpu       up 3-00:00:00    9  mix  gpu[002-010]
hm       up 3-00:00:00    1  mix  hm034
hm       up 3-00:00:00   36  alloc hm[002-033,036-039]
hm       up 3-00:00:00    1  idle hm035
```

Figure 8 - Output of sinfo command

## Submit the job

We can consider three cases of submitting a job

### 1. Submitting a simple standalone job

This is a simple submit script which is to be submitted

```
$ sbatch slurm-job.sh
Submitted batch job 106
```

### 2. Submit a job that's dependent on a prerequisite job being completed

Consider a requirement of pre-processing a job before proceeding to actual processing. Pre-processing is generally done on a single core. In this scenario, the actual processing script is dependent on the outcome of pre-processing script.

here's a simple job script. Note that the Slurm -J option is used to give the job a name.

```
#!/usr/bin/env bash
#SBATCH -p standard
#SBATCH -J simple
sleep 60
```

```
Submit the job: $ sbatch simple.sh
Submitted batch job 149
```

Now we'll submit another job that's dependent on the previous job. There are many ways to specify the dependency conditions, but the "singleton" method is the simplest. The Slurm `-d singleton` argument tells Slurm not to dispatch this job until all previous jobs with the same name have completed.

```
$ sbatch -d singleton simple.sh //may be used for first
pre-processing on a core and then submitting
Submitted batch job 150
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 150 standard simpleuser1 PD 0:00 1 (Dependency)
 149 standard simpleuser1 R 0:17 1 atom01
```

Once the prerequisite job finishes the dependent job is dispatched.

```
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 150 standard simpleuser1 R 0:31 1 atom01
```

### 3. Submit a job with a reservation allocated

Slurm has the ability to reserve resources for jobs being executed by select users and/or select bank accounts. A resource reservation identifies the resources in that reservation and a time period during which the reservation is available. The resources which can be reserved include cores, nodes.

Use the command given below to check the reservation name allocated to your user account

```
$ scontrol show reserv
```

If your 'user account' is associated with any reservation the above command will show you the same. For eg. The reservation name given is `user_11`. Use the command given below to make use of this reservation

```
$ sbatch --reservation=user_11 simple.sh
```

#### 4. Submitting multiple jobs with minor or no changes (array jobs)

A **SLURM job array** is a collection of jobs that differs from each other by only a single index parameter. Job arrays can be used to submit and manage a large number of jobs with similar settings.

```
# Submit a job array with index values between 0 and 31
$ sbatch --array=0-31 -N1 tmp

# Submit a job array with index values of 1, 3, 5 and 7
$ sbatch --array=1,3,5,7 -N1 tmp

# Submit a job array with index values between 1 and 7
# with a step size of 2 (i.e. 1, 3, 5 and 7)
$ sbatch --array=1-7:2 -N1 tmp
```

Figure 9 – Snapshot depicting the usage of “Job Array”

N1 is specifying the number of nodes you want to use for your job. example: N1 -one node, N4 - four nodes. Instead of tmp here you can use the below example script.

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --ntasks-per-node=48
#SBATCH --error=job.%A_%a.err
#SBATCH --output=job.%A_%a.out
#SBATCH --time=01:00:00
#SBATCH --partition=standard

module load compiler/intel/2018.2.199
cd /home/guest/
export OMP_NUM_THREADS=${SLURM_ARRAY_TASK_ID}
/home/guest/ /md_omp
```

#### List jobs

Monitoring jobs on SLURM can be done using the command **squeue**. squeue is used to view job and job step information for jobs managed by SLURM.

```
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
 106 standard slurm-jo user1 R 0:04 1 atom01
```

## Get job details

**scontrol** can be used to report more detailed information about nodes, partitions, jobs, job steps, and configuration.

**scontrol show node** - shows detailed information about compute nodes.

```
[root@master ~]# scontrol show node cn001
NodeName=cn001 Arch=x86_64 CoresPerSocket=24
CPUAlloc=0 CPUTot=48 CPULoad=0.01
AvailableFeatures=cpu,centos7
ActiveFeatures=cpu,centos7
Gres=(null)
NodeAddr=cn001 NodeHostName=cn001 Version=19.05.5
OS=Linux 3.10.0-1062.9.1.el7.x86_64 #1 SMP Fri Dec 6 15:49:49 UTC 2019
RealMemory=1 AllocMem=0 FreeMem=187048 Sockets=2 Boards=1
State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
Partitions=standard,debug
BootTime=2020-03-30T19:53 SlurmdStartTime=2020-03-30T19:20:32
CfgTRES=cpu=48,mem=1M,billing=48
AllocTRES=
CapWatts=n/a
CurrentWatts=0 AveWatts=0
ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Figure 10 – scontrol show node displays compute node information

**scontrol show partition** - shows detailed information about a specific partition

```
[root@master ~]# scontrol show partition hm
PartitionName=hm
AllowGroups=ALL AllowAccounts=ALL AllowQos=ALL
AllocNodes=ALL Default=NO QoS=N/A
DefaultTime=NONE DisableRootJobs=NO ExclusiveUser=NO GraceTime=0 Hidden=NO
MaxNodes=UNLIMITED MaxTime=3-00:00:00 MinNodes=0 LLN=NO MaxCPUsPerNode=UNLIMITED
Nodes=hm[002-039]
PriorityJobFactor=50 PriorityTier=50 RootOnly=NO ReqResv=NO OverSubscribe=NO
OverTimeLimit=NONE PreemptMode=OFF
State=UP TotalCPUs=1824 TotalNodes=38 SelectTypeParameters=NONE
JobDefaults=(null)
DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

Figure 11 – scontrol show partition displays specific partition details

**scontrol show job** - shows detailed information about a specific job or all jobs if no job id is given.

```
[root@master ~]# scontrol show job 6703
JobId=6703 JobName=lammps
  UserId=(b) GroupId=(b) MCS_label=N/A
  Priority=21895 nice=0 Account=nsmapp QoS=normal
  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=04:38:27 TimeLimit=08:51:00 TimeMin=N/A
  SubmitTime=2020-04-14T17:20:47 EligibleTime=2020-04-14T17:20:47
  AccrueTime=2020-04-14T17:20:47
  StartTime=2020-04-14T17:20:48 EndTime=2020-04-15T02:11:48 Deadline=N/A
  SuspendTime=None SecsPreSuspend=0 LastSchedEval=2020-04-14T17:20:48
  Partition=standard AllocNode:Sid=login01:96927
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=cn[049-069],hm[019-029]
  BatchHost=cn049
  NumNodes=32 NumCPUs=1280 NumTasks=1280 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
  TRES=cpu=1280,node=32,billing=1280
  Socks/Node=* NtasksPerN:B:S:C=40:0:*:* CoreSpec=*
  MinCPUsNode=40 MinMemoryNode=0 MinTmpDiskNode=0
  Features=(null) DelayBoot=00:00:00
  OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
  Command=/home/.../NEW_LAMMPS/lammps-7Aug19/bench/lammps_cpu_mpi_32_node.sh
  WorkDir=/home/.../NEW_LAMMPS/lammps-7Aug19/bench
  StdErr=/home/.../NEW_LAMMPS/lammps-7Aug19/bench/job.%J.err_32_node_40
  StdIn=/dev/null
  StdOut=/home/.../NEW_LAMMPS/lammps-7Aug19/bench/job.%J.out_32_node_40
  Power=
```

Figure 12 – scontrol show job displays specific job information

**scontrol update job** - change attributes of the submitted job. like time limit, priority (root only)

```
$ scontrol show job 106
JobId=106 Name=slurm-job.sh
  UserId=user1(1001) GroupId=user1(1001)
  Priority=4294901717 Account=(null) QOS=normal
  JobState=RUNNING Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
  RunTime=00:00:07 TimeLimit=14-00:00:0 TimeMin=N/A
  SubmitTime=2013-01-26T12:55:02 EligibleTime=2013-01-26T12:55:02
  StartTime=2013-01-26T12:55:02 EndTime=Unknown
  PreemptTime=None SuspendTime=None SecsPreSuspend=0
  Partition=standard AllocNode:Sid=atom-head1:3526
  ReqNodeList=(null) ExcNodeList=(null)
  NodeList=atom01
  BatchHost=atom01
  NumNodes=1 NumCPUs=2 CPUs/Task=1 ReqS:C:T=*:*:~*
  MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
  Features=(null) Gres=(null) Reservation=(null)
  Shared=0 Contiguous=0 Licenses=(null) Network=(null)
  Command=/home/user1/slurm/local/slurm-job.sh
  WorkDir=/home/user1/slurm/local
```

**scontrol update job= 106 TimeLimit=15-00:00:0**

**Suspend a job (root only):**

```
# scontrol suspend 135
# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
```

```
135 standard simple.s user1 S 0:10 1 atom01
```

### Resume a job (root only):

```
# scontrol resume 135
# squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
135 standard simple.s user1 R 0:13 1 atom01
```

### Kill a job. Users can kill their own jobs, root can kill any job.

```
$ scancel 135
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
```

### Hold a job:

```
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
139 standard simple user1 PD 0:00 1 (Dependency)
138 standard simple user1 R 0:16 1 atom01
$ scontrol hold 139
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
139 standard simple user1 PD 0:00 1 (JobHeldUser)
138 standard simple user1 R 0:32 1 atom01
```

### Release a job:

```
$ scontrol release 139
$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
139 standard simple user1 PD 0:00 1 (Dependency)
138 standard simple user1 R 0:46 1 atom01
```

# Addressing Basic Security Concerns

---

Your account on PARAM Yukti is 'private to you'. You are responsible for any actions emanating from your account. It is suggested that you should never share the password with anyone including your friends and system administrators!!

Please note that, by default, a new account created on PARAM Yukti is readable by everyone on the system. The following simple commands will make your account adequately safe.

<b>chmod 700 /home/\$user</b>	! will ensure that only yourself can read, write and ! execute files in your home directory
<b>chmod 750 /home/\$user</b>	! will enable yourself and the members of your ! group to read and execute files in your home ! directory
<b>chmod 755 /home/\$user</b>	! will enable yourself, your group members and ! everyone else to read and execute files in your ! directory
<b>chmod 777 /home/\$user</b>	! will enable everyone on the system to read, ! write and execute files in your home directory. ! This is a sort of 'free for all' situation. This ! should be used very judiciously



# More about Batch Jobs (SLURM)

SLURM (Simple Linux Utility for Resource Management) is a workload manager that provides a framework for job queues, allocation of compute nodes, and the start and execution of jobs.

## It is important to note:

- Compilations are done on the login node. Only the execution is scheduled via SLURM on the compute nodes
- Upon Submission of a Job script, each job gets a unique Job Id. This can be obtained from the '**squeue**' command.
- The Job Id is also appended to the output and error filenames.

## Parameters used in the SLURM job script

The job flags are used with SBATCH command. The syntax for the SLURM directive in a script is "#SBATCH <flag>". Some of the flags are used with the srun and salloc commands.

Resource	Flag Syntax	Description
<b>partition</b>	--partition=partition name	Partition is a queue for jobs.
<b>time</b>	--time=01:00:00	Time limit for the job.
<b>nodes</b>	--nodes=2	Number of compute nodes for the job.
<b>cpus/core s</b>	--ntasks-per-node=8	Corresponds to number of cores on the compute node.
<b>resource feature</b>	--gres=gpu:2	Request use of GPUs on compute nodes
<b>account</b>	--account=group-slurm-account	Users may belong to groups or accounts.
<b>job name</b>	--job-name="lammeps"	Name of job.
<b>output file</b>	--output=lammeps.out	Name of file for stdout.
	-w, --nodelist	Request a specific list of hosts.

--mail-type= Notify user by email when certain event types occur. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL TIME\_LIMIT, TIME\_LIMIT\_90 (reached 90 percent of time limit), TIME\_LIMIT\_80 (reached 80 percent of time limit), and TIME\_LIMIT\_50 (reached 50

Resource	Flag Syntax	Description
<b>email address</b>	--mail-user username@jncasr.ac.in User to receive email notification of state changes as defined by --mail-type	percent of time limit). Multiple type values may be specified in a comma separated list User's email address
<b>access</b>	--exclusive	Exclusive access to compute nodes. The job allocation cannot share nodes with other running jobs

## Script for a Sequential Job

```
#!/bin/bash
#SBATCH -N 1 // number of nodes
#SBATCH --ntasks-per-node=1 // number of cores per node
#SBATCH --error=job.%J.err // name of output file
#SBATCH --output=job.%J.out // name of error file
#SBATCH --time=01:00:00 // time required to execute the program #SBATCH
--partition=standard // specifies queue name (standard is the default
partition if you does not specify any partition job will be
submitted using default partition). For other partitions you can specify hm
or gpu

// To load the module //

module load compiler/intel/2018.2.199

cd <Path of the executable>

/home/cdac/a.out {Name of the executable}
```

## Script for a Parallel OpenMP Job

```
#!/bin/bash
#SBATCH -N 1 // Number of nodes
#SBATCH --ntasks-per-node=48 // Number of core per node
#SBATCH --error=job.%J.err // Name of output file
#SBATCH --output=job.%J.out // Name of error file
#SBATCH --time=01:00:00 // Time take to execute the program #SBATCH --
partition=standard // specifies queue name(standard is the default partition
```

```

cd <path of the
executable> or
cd $SLURM_SUBMIT_DIR //To run job in the directory from where it
is submitted

export OMP_NUM_THREADS=48 (Depending upon your requirement you can
change number of threads. If total number of threads per node is more
than 48, multiple threads will share core(s) and performance may
degrade)

/home/  c/a.out (Name of the executable)

```

## Script for Parallel Job – MPI (Message Passing Interface)

```

#!/bin/sh

#SBATCH -N 16 // Number of nodes
#SBATCH --ntasks-per-node=48 // Number of cores per node #SBATCH
--time=06:50:20 // Time required to execute the program #SBATCH
--job-name=lammps // Name of application
#SBATCH --error=job.%J.err_16_node_48 // Name of the output file
#SBATCH --output=job.%J.out_16_node_48 // Name of the error file
#SBATCH --partition=standard // Partition or queue name

// To load the module //
module load compiler/intel/2018.2.199

// Below are Intel MPI specific settings //
export I_MPI_FALLBACK=disable

export I_MPI_FABRICS=shm:dapl
export I_MPI_DEBUG=9 // Level of MPI verbosity //

cd $SLURM_SUBMIT_DIR
or
cd /home/ /LAMMPS_2018COMPILER/lammps-22Aug18/bench

// Command to run the lammps in Parallel //

time mpiexec.hydra -n $SLURM_NTASKS -genv OMP_NUM_THREADS 1
/home/ /LAMMPS_2018COMPILER/lammps-22Aug18/src/lmp_intel_cpu_intelmpi
-in in.lj

```

## Script for Hybrid Parallel Job – (MPI + OpenMP)

```

#!/bin/sh

#SBATCH -N 16 // Number of nodes
#SBATCH --ntasks-per-node=48 // Number of cores for node #SBATCH
--time=06:50:20 // Time required to execute the program #SBATCH
--job-name=lammps // Name of application
#SBATCH --error=job.%J.err_16_node_48 // Name of the output file
#SBATCH --output=job.%J.out_16_node_48 // Name of the error file
#SBATCH --partition=standard // Partition or queue name

cd $SLURM_SUBMIT_DIR

```

```
// To load the module //
module load compiler/intel/2018.2.199
```

```
// Below are Intel MPI specific settings //
export I_MPI_FALLBACK=disable
export I_MPI_FABRICS=shm:dapl
```

```
export I_MPI_DEBUG=9 // Level of MPI verbosity //
export OMP_NUM_THREADS=24 //Possibly then total no. of MPI ranks will be =
(total no. of cores, in this case 16 nodes x 48 cores/node) divided by (no.
of threads per MPI rank i.e. 24)

// Command to run the lammps in Parallel //
time mpiexec.hydra -n 32 lammps.exe -in
```

```
in.lj
```

## I am familiar with PBS/ TORQUE. How do I migrate to SLURM?

Environment Variables	PBS/Torque	SLURM
Job Id	\$PBS_JOBID	\$SLURM_JOBID
Submit Directory	\$PBS_JOBID	\$SLURM_SUBMIT_DIR
Node List	\$PBS_NODEFILE	\$SLURM_JOB_NODELIST
Job Specification	PBS/Torque	SLURM
Script directive	#PBS	#SBATCH
Job Name	-N [name]	--job-name=[name] OR -J [name]
Node Count	-l nodes=[count]	--nodes=[min[-max]] OR -N [min[-max]]
CPU count	-l ppn=[count]	---ntasks-per-node=[count]
CPUs Per Task		--cpus-per-task=[count]
Memory Size	-l mem-[MB]	--mem=[MB] OR – mem_per_cpu=[MB]
Wall Clock Limit	-l walltime=[hh:mm:ss]	--time=[min] OR – mem_per_cpu=[MB]
Node Properties	-l nodes=4.ppn=8:[property]	--constraint=[list]
Standard Output File	-o [file_name]	--output=[file_name] OR -o _ [file_name]

---

Standard Error File	-e [file_name]	--error=[file_name] OR -e {file_name}
Combine stdout/stderr	-j oe (both to stdout)	(This is default if you do not specify --error)
Job Arrays	-t [array_spec]	--array=[array_spec] OR -a [array_spec]
Delay Job Start	-a [time]	--begin=[time]

---

# Preparing Your Own Executable

---

The compilations are done on the login node, whereas the execution happens on the compute nodes via the scheduler (SLURM).

**Note:** The Compilation and execution must be done with same libraries and matching version to avoid unexpected results.

## Steps:

1. Load required modules on the login node.
2. Do the compilation.
3. Open the job submission script and specify the same modules to be loaded as used while compilation.
4. Submit the script.

The directory contains a few sample programs and their sample job submission scripts. The compilation and execution instructions are described in the beginning of the respective files.

The user can copy the directory to his/her home directory and further try compiling and executing these sample codes. The command for copying is as follows:

```
cp -r /home/apps/Docs/samples/ ~/.
```

1. mm.c - Serial Version of Matrix-Matrix Multiplication of two NxN matrices
2. mm\_omp.c - Basic OpenMP Version of Matrix-Matrix Multiplication of two NxN matrices
3. mm\_mpi.c - Basic MPI Version of Matrix-Matrix Multiplication of two NxN matrices
4. mm\_acc.c - OpenAcc Version of Matrix-Matrix Multiplication of two NxN matrices
5. mm\_blas.cu - CUDA Matrix Multiplication program using the cuBlas library.
6. mm\_mkl.c - MKL Matrix Multiplication program.
7. laplace\_acc.c - OpenACC version of the basic stencil problem.

It is recommended to use the intel compilers since they are better optimized for the hardware.

## Compilers

Compilers	Description	Versions Available
gcc/gfortran	GNU Compiler (C/C++/Fortran)	4.8.5, 5.5.0, 7.3.0, 8.3.0, 9.3.0
icc/icpc/fort	Intel Compilers (C/C++/Fortran)	16.x, 17.x, 18.x, 19.x
mpicc/mpicxx/mpif90	Intel-MPI with GNU compilers (C/C++/Fortran)	16.x, 17.x, 18.x, 19.x
mpiicc/mpiicpc/mpiifort	Intel-MPI with Intel compilers (C/C++/Fortran)	16.x, 17.x, 18.x, 19.x
nvcc	CUDA C Compiler	7.5, 8.0, 9.0, 9.2, 10.0, 10.1, 10.2
pgcc/pgc++/pgfortran	PGI Compiler (C/C++/Fortran)	19.4, 19.10

## Optimization Flags

Optimization flags are meant for uniprocessor optimization, wherein, the compiler tries to optimize the program, on the basis of the level of optimization. The optimization flags may also change the precision of output produced from the executable. The optimization flags can be explored more on the respective compiler pages. A few examples are given below.

```
Intel:      -O3
-xHost GNU: -O3
PGI: -fast
```

Given next is a brief description of the compilation and execution of the various types of programs. However, for certain bigger applications, the loading of additional dependency libraries might be required.

## C Program:

```
Setting up of environment: module load compiler/intel/2018.2.199
compiler/gcc/7.3.0
compilation: icc -O3 -xHost <<prog_name.c>>
Execution: ./a.out
```

## C + OpenMP Program:

```
Setting up of environment:  module load
compiler/intel/2018.2.199 compiler/gcc/7.3.0
Compilation: icc -O3 -xHost -qopenmp <<prog_name.c>>
Execution: ./a.out
```

## C + MPI Program:

```
Setting up of environment:  module load
compiler/intel/2018.2.199 compiler/gcc/7.3.0
Compilation: mpiicc -O3 -xHost <<prog_name.c>>
Execution: mpirun -n <<num_procs>> ./a.out
```

## C + MKL Program:

```
Setting up of environment:
module load compiler/intel/2018.2.199 compiler/gcc/7.3.0
Compilation: icc -O3 -xHost -mkl <<prog_name.c>>
Execution: ./a.out
```

## CUDA Program:

```
Setting up of environment:
module load compiler/cuda/10.1 compiler/gcc/7.3.0
```

### Example (1)

```
Compilation: nvcc -arch=sm_70 <<prog_name.cu>>
Execution: ./a.out
```

Note: The optimization switch `-arch=sm_70` is intended for Volta V100 GPUs and is valid for CUDA 9 and later. Similarly, older versions of CUDA have compatibility with lower versions of GCC only. Accordingly, appropriate modules of GCC must be loaded.

### Example (2)

```
Compilation: nvcc -arch=sm_70
/home/apps/Docs/samples/mm_blas.cu lcublas
Execution: ./a.out
```

## CUDA + OpenMP Program:

```
Setting up of environment:
module load compiler/cuda/10.1 compiler/gcc/7.3.0
```

### Example (1)

```
Compilation: nvcc -arch=sm_70 -Xcompiler="-fopenmp" -lgomp
/home/apps/Docs/samples/mm_blas_omp.cu -lcublas
Execution: ./a.out
```

### Example (2)



```

Compilation: g++ -fopenmp
/home/apps/Docs/samples/mm_blas_omp.c
I/opt/ohpc/pub/apps/cuda/cuda-10.1/include
L/opt/ohpc/pub/apps/cuda/cuda-10.1/lib64 -lcublas
Execution: ./a.out

```

## OpenACC Program:

```

Setting up of environment:
module load compiler/pgi/19.10      compiler/cuda/10.1

Compilation for GPU: pgcc -acc -fast -Minfo=all -ta=tesla:cc70,managed
/home/apps/Docs/samples/laplace_acc.c
Execution: ./a.out

Compilation for CPU: pgcc -acc -fast -Minfo=all -ta=multicore -
tp=skylake /home/apps/Docs/samples/laplace_acc.c
Execution: ./a.out

```

## Job Submission on Scheduler (SLURM)

A sample job submission scripts for each of the sample programs is given. Upon completion/termination of the execution, two files (output and error) are generated.

A few sample commands for SLURM are as follows:

<pre>sinfo</pre>	Lists out the status of resources in the system
<pre>squeue</pre>	Lists out the Job information in the system
<pre>sbatch &lt;&lt;job_script&gt;&gt;</pre>	Submitting a job to the scheduler
<pre>scancel &lt;&lt;job_name&gt;&gt;</pre>	Delete a job

# Job Scheduling on PARAM Yukti

## Scheduler

PARAM Yukti has Slurm-19.05.0-1 (open source) as a workload manager for the HPC facility. Slurm is a widely used batch scheduler in top500 HPC list. PARAM Yukti consists of three types of compute nodes: i.e. CPU only (192 GB) nodes, High memory (768 GB) nodes and Nvidia GPGPU (192 GB) enabled.

Following partitions/queues have been defined for different requirements

1. **standard**: CPU, High memory and GPU Jobs
2. **gpu**: GPU and CPU jobs
3. **hm**: High memory-intensive jobs

All users can submit to the Standard partition. The standard Partition contains CPU, high memory, and GPU nodes. GPU partition contains only GPU nodes. If the user wants to submit a job only on GPU nodes, he/she can use GPU partition. If the user wants to submit a job only on high memory, he/she can use hm partition.

**Note:**User has to specify `#SBATCH -gres=gpu:1/2` in their job script if user wants to use 1 or 2 GPU cards on GPU nodes

## sinfo

This Slurm command is used to view available **partition** and **node** information on the cluster.

```

PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
standard* up 3-00:00:00      3  maint cn[108-110]
standard* up 3-00:00:00     21  mix  cn[080-089],gpu[001-010],hm034
standard* up 3-00:00:00    131  alloc cn[001-043,045-079,090-098,100-101,103-107],hm[001-033,036-039]
standard* up 3-00:00:00      4  idle  cn[044,099,102],hm035
gpu       up 3-00:00:00      9  mix  gpu[002-010]
hm       up 3-00:00:00      1  mix  hm034
hm       up 3-00:00:00     36  alloc hm[002-033,036-039]
hm       up 3-00:00:00      1  idle  hm035

```

Figure 13 – sinfo Command

## walltime

Walltime parameter defines as to how long your job will run. The maximum runtime of a job allowed as per the QoS Policy. If more than 3 days are required, a special request needs to be sent to HPC coordinator and it will be dealt with on a case to case basis. The command line to specify walltime is given below.

```
srun -t walltime <days-hours:mins:seconds>
```

and also as part of the submit scripts described in the manual. If a job does not get completed within the walltime specified in the script, it will get terminated.

The biggest advantage of specifying appropriate walltime is that the efficiency of scheduling improves resulting in improved throughput in all jobs including yours. You are encouraged to arrive at the appropriate walltime for your job by executing your jobs few times.

---

**NOTE:** You are requested to explicitly specify the walltime in your command lines and scripts.

---

### Per user

- Every user will have a quota of 650 GB of soft limit and 1TB of hard limit with a grace period of 7 days in the HOME file system (/home) and 2TB of soft limit and 4TB of the hard limit with a grace period of 14 days in SCRATCH file system
- Users are recommended to copy their execution environment and input files to the scratch file system (/scratch/<username>) during job running and copy output data back to the HOME area
- File retention policy has been implemented on Lustre storage for the "/scratch" file system. As per the policy, any files that have not been accessed for the last 3 months will be deleted permanently
- Three QoS (Quality of services ) are created according to different job sizes and wall time. Resource limits for users are defined as per below QoS policy

### QoS policy

#### 1. Small QoS:

Maximum No.of.  
nodes= 4 wall time=3  
days  
Maximum running jobs per user at a  
time=5 priority=0

---

**Note:** This is the default QoS, no need to mention this QoS in the job script.

---

#### 2. Medium QoS:

Maximum No.of.  
Nodes=16 Wall time=2  
days

---

**Note:** To use this QoS, you must mention this QoS in the job script while submitting the job

Maximum running jobs per user at a time =3 priority=50(more than small QoS)

Example for using this QoS in the job script #SBATCH --qos=medium

### 3. Large QoS:

Maximum No.of.

Nodes=32 wall time=1

day

Maximum running jobs per user at a time=1 priority=100(more than small and medium)

---

**Note:** To use this QoS, you must mention these QoS in the job script while submitting the job

Example for using this QoS in a job script

#SBATCH --qos=large

---

**Note:** QoS policy only applicable to JNCASR internal users.

---

## Scheduling Type

PARAM Yukti has been configured with Slurm's backfill scheduling policy. It is good for ensuring higher system utilization; it will start lower priority jobs if doing so does not delay the expected start time of any higher priority jobs. Since the expected start time of pending jobs depends upon the expected completion time of running jobs, reasonably accurate time limits are important for backfill scheduling to work well.

## Job Priority

The job's priority at any given time will be a weighted sum of all the factors that have been enabled in the slurm.conf file. Job priority can be expressed as:

```
Job_priority =
  (PriorityWeightAge) * (age_factor) +
  (PriorityWeightFairshare) * (fair-share_factor)
+ (PriorityWeightJobSize) * (job_size_factor) +
  (PriorityWeightPartition) * (partition_factor) +
  (PriorityWeightQOS) * (QOS_factor) +
  SUM(TRES_weight_cpu * TRES_factor_cpu,
      TRES_weight_<type> * TRES_factor_<type>,
      ...)
```

All of the factors in this formula are floating-point numbers that range from 0.0 to 1.0. The weights are unsigned 32-bit integers. The job's priority is an integer that ranges between 0 and 4294967295. The larger the number, the higher the job will be positioned in the queue, and the sooner the job will be scheduled. A job's priority, and hence its order in the queue, can vary over time. For example, the longer a job sits in the queue, the higher its priority will grow when the age weight is non-zero.

**Age Factor:** The age factor represents the length of time a job has been sitting in the queue and eligible to run. The current value for Age factor is 10000.

**Job Size Factor:** The job size factor correlates to the number of nodes or CPUs the job has requested. The current value for the Job Size factor is 1000.

**Partition Factor:** Each node partition can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request to run in this partition. The current value for the partition factor is 15000.

**Quality of Service (QOS) Factor:** Each QOS can be assigned an integer priority. The larger the number, the greater the job priority will be for jobs that request this QOS. The current value for QOS factor is 100000.

**Fair-share Factor:** The fair-share component to a job's priority influences the order in which a user's queued jobs are scheduled to run based on the portion of the computing resources they have been allocated and the resources their jobs have already consumed. The current value for fair-share factor is 100000.

## SSHARE

This tool is for listing the shares of association to a cluster.

```

root@login01 ~]# sshare
Account      User  RawShares  NormShares  RawUsage  EffectvUsage  FairShare
-----
root         root      1      0.000000    25702864    1.000000
root         root      1      0.009804    17216646    0.669834    0.038462
jncasr      jncasr    60     0.588235      0      0.000000

```

Figure 14 - Listing the shares of association to a cluster

## ACCOUNTING

The accounting system tracks and manages HPC resource usage. As jobs are completed or resources are utilized, accounts are charged and resource usage is recorded. Accounting the policy is like a bank/Credit System, where each department can be allocated with some pre-defined budget on a quarterly basis for CPU usage. As and when the resources are utilized, the amount will be deducted. The allocation will be reset at end of every quarter.

### sacct

This command can report resource usage for running or terminated jobs including

individual tasks, which can be useful to detect load imbalance between the tasks.

**sstat**

This command can be used to status only currently running jobs.

**sreport**

This command can be used to generate reports based upon all jobs executed in a particular time interval.

# Debugging Your Codes

---

## Introduction

A **debugger** or **debugging tool** is a computer program that is used to test and debug other programs (the "target" program).

When the program "traps" or reaches a preset condition, the debugger typically shows the location in the original code if it is a source-level debugger or symbolic debugger, commonly now seen in **integrated development environments**.

Debuggers also offer more sophisticated functions such as running a program step by step (single-stepping or program animation), stopping (breaking) (pausing the program to examine the current state) at some event, or specified instruction by means of a breakpoint and tracking the values of variables.

Some debuggers have the ability to modify the program state while it is running. It may also be possible to continue execution at a different location in the program to bypass a crash or logical error.

## Basics How-Tos

### Compilation

Compilation with a separate flag '-g' is required since the program needs to be linked with debugging symbols.

```
gcc -g <program_name.c>  
e.x. gcc -g random_generator.c
```

### Running with gdb:

gdb is a command-line utility available with almost all Linux systems' compiler collection packages.

```
gdb <executable.out>  
e.x. gdb a.out
```

**Basic gdb commands (to be executed in gdb command line window):****Start:**

Starts the program execution and stops at the first line of the main procedure. Command-line arguments may be provided if any.

**Run:**

Starts the program execution but does not stop. It stops only when any error or program trap occurs. Command-line arguments may be provided if any.

**Help:**

Prints the list of commands available. Specifying 'help' followed by a command (e.x. 'help run') displays more information about that command.

**File <filename>:**

Loads a binary program that is compiled with '-g' flag for debugging.

**List [line\_no]**

Displays the source code (nearby 10 lines) of the program in the execution where the execution stopped. If 'line\_no' is specified, it displays the source code (10 lines) at the specified line.

**Info:**

Displays more information about the set of utilities and saved information by the debugger. For example; 'info breakpoints' will list all the breakpoints, similarly 'info watchpoints' will list all the watchpoints set by the user while debugging their programs.

**Print <expression>:**

Prints the values of variables/expressions at the currently running instance of the program.

**Step N:**

Steps the program one (or 'N') instructions ahead or till the program stops for any reason. Steps through each and every instruction even if it is function call (only function or instruction compiled with debugging flags).

**next:**

This command also steps through the instructions of the program. Unlike 'step' command, if the current source code line calls a subroutine, this command does not enter the subroutine, but instead steps over the call, in effect treating it as a single



source line.

**Continue:**

This command continues the stopped program till the next breakpoint has occurred or till the end of the program. It is used to continue from a paused/debug point state.

**Break [sourcefile:]<line\_no> [if condition]:**

Stops the program at the specified line number and provides a breakpoint for the user. Specific source code file and breakpoints based on a condition can also be set for specific cases. You can also view the list of breakpoints set, by using the 'info breakpoints' command.

**watch <expression>:**

A watchpoint means to break the program or stop the execution of the program when the value of the expression provided is changed. Using the watch command-specific variables can be watched for value changes. You can also view the list of watchpoints by using the 'info watchpoints' command.

**Delete <breakpoint number>**

Delete command deletes a breakpoint or a watchpoint that has been set by a user while debugging the program.

**Backtrace:**

Prints the backtrace of all stack frames of the program. Provides the call stack and more other information about the running program.

These are some of the most powerful utilities that can be used to debug your programs using gdb. gdb is not limited to these commands and contains a rich set of features that can allow you to debug multi-threaded programs as well. Also, all the commands, along with the ones listed above have 'n' number of different variants for more in-depth control. The same can be utilized using the help page of gdb.

**Using gdb (example – inspecting the code)**

For this case study, we have a small program that generates a long unique random number for each run.

Let's look at the code we have.

```

#include <stdio.h>           //printf
#include <stdlib.h>         //malloc, srand, rand
#include <unistd.h>         //getpid

#define N 100
#define N_LEN 100

//Generate a short random number
short rand_fract(void) {
    short sum = 1;
    for (short i = 0; i < (rand() % N); ++i)
        for (short j = 0; i < N; ++j) {
            int value = (i * j) / (i + j);
            sum += (value != 0) ? value : sum;
        }
    return sum;
}

//Returns the factorial of a number
long long factorial(unsigned int x) {
    if (x == 1 || x == 0)
        return 1LL;
    else
        return (x * factorial(x - 1));
}

```

Figure 15 – Snapshot of debugging process

Things to note:

- 1) We have a few libraries included for the functions that are used in the program.
- 2) We have two '#define' statements:
  - a. 'N' for the number of times the 'rand\_fract' function will spend in calculating the random number.
  - b. 'N\_LEN' for the length of the final random number string generated. Currently, it is set to '100' which means that the long random number will be of length 100.
- 3) Then, we have a function by the name 'rand\_fract' that iterates over two loops and using the values of iterators ('i' and 'j'), it calculates a small random number. Since 'rand()' function is used for the outer loop, its number of iterations cannot be clearly defined which gives the function a random nature.
- 4) The next function is as simple as its name is. It just takes an unsigned integer and returns its factorial.

## PART 2:

```

int main (int argc, char *argv[]) {

    short f1 = 0;

    //Create a random seed based on process id.
    srand((unsigned int) getpid());

    //Generate a random number salt.
    f1 = rand_fract() % 10;

    //Get the factorial of the number
    long long random_fact = factorial(f1);

    //Normalize the factorial to number modulo N_LEN + 1
    int normalized_fact = random_fact % (N_LEN + 1);

    int *array = NULL;

    //Create an array of size obtained from normalized factorial modulo N_LEN + 1
    array = (int *) malloc (sizeof (int) * normalized_fact);
    if (array == NULL) { printf("Not enough memory\n"); return -1; }

    //Populate the array with integers ni reverse order
    //Double the number five times if it is even
    for (int i = 0; i < normalized_fact; ++i) {
        array[i] = (normalized_fact - i);
    }

    //Print the serial number
    for (int i = normalized_fact - 1; i >= 0; --i)
        printf("%0d", (array[i] + rand()) % 10);
    for (int i = (N_LEN - normalized_fact); i > 0; --i)
        printf("%0d", (rand() % 10));
    printf("\n");

    //Free allocated memory
    free(array);

    return 0;
}

```

Figure 16 – Snapshot of debugging process

## Things to note:

- 1) This is the main function of the program.
- 2) The flow of the main function is as follows:
  - a. The program first sets a random seed using the process-id of the program.
  - b. It calls 'rand\_fract' function and the resultant random number is operated by a modulo 10 operation. Finally, the result is stored in the variable 'f1'.
  - c. Next, the factorial of the obtained 'f1' is calculated and stored in 'random\_fact'.
  - d. This result is again passed through a modulo 'N\_LEN + 1' and stored in 'normalized\_fact'.
  - e. Then a dynamic array is constructed and partially filled with integer values in descending order from the 'normalized\_fact' value.

- f. Finally, the partial array is printed by mixing the value of the array with `rand()` function values followed by a modulo 10 operation.
- g. The remaining partial part of the final random value is generated using a basic `rand()` modulo 10 operation.

### Using gdb (example – using the debugger)

The code that we looked upon seems correct, as well as it compiles successfully without any errors. But, when we run this code snippet, this is the result we get.

```
$ gcc random_generator.c
$ ./a.out
Floating point exception (core dumped)
$ █
```

Figure 17- Output at a debugging stage

The program ended up with a core dump without giving much information but just a 'Floating point exception'. Now let's compile the code with debugging information and run the program simply with gdb.

```
$ gcc -g random_generator.c
$ gdb a.out
GNU gdb (GDB) Fedora 8.3.50.20190824-25.fc31
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) set style enabled off
(gdb) run
Starting program: /home/vineetm/debugger/a.out

Program received signal SIGFPE, Arithmetic exception.
0x00000000004011cc in rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) █
```

Figure 18 – Snapshot of debugging process

Here we compiled the code using '-g' and then used the 'run' command we studied earlier for running the program. You can observe that the debugger stopped at line number 13 where the 'Floating point exception (SIGFPE)' occurred. At this point, we can even go and check the code at line number 13. But for now, let's check what other information we can get from the debugger. Let's check the values of the variables 'i' and 'j' at this point.

```
(gdb) run
Starting program: /home/██████████/debugger/a.out

Program received signal SIGFPE, Arithmetic exception.
0x0000000004011cc in rand_fract () at random_generator.c:13
13                               int value = (i * j) / (i + j);
(gdb) print i
$1 = 0
(gdb) print j
$2 = 0
(gdb) █
```

Figure 19 – Output depicting “Arithmetic Exception”

The values of both 'i' and 'j' appear to be '0' and thus a **divide by zero** exception is what caused our program to terminate. Let's update the code such that the value of 'i' and 'j' will never become '0'. This is the modified code:

```
//Generate a short random number
short rand_fract(void) {
    short sum = 1;
    for (short i = 1; i < (rand() % N); ++i)
        for (short j = 1; i < N; ++j) {
            int value = (i * j) / (i + j);
            sum += (value != 0) ? value : sum;
        }
    return sum;
}
```

Figure 20 – Snapshot of debugging process

Thus, we just updated the loop index variables to start from '1' instead of '0'. **Thus, using gdb, it was very simple to identify the point where the error occurred.** Let's re-run our updated code and check what we get.

```
$ gcc random_generator.c
$ ./a.out
Floating point exception (core dumped)
$ █
```

Figure 21 – Well, we dumped core !!

WHAT!? This is unexpected. We just cured the error part of our program and still getting an FPE. Let's go through the debugger and check where the error point is right now.

```
$ gcc -g random_generator.c
$ gdb a.out
GNU gdb (GDB) Fedora 8.3.50.20190824-25.fc31
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) set style enabled off
(gdb) run
Starting program: /home/████████/bugger/a.out

Program received signal SIGFPE, Arithmetic exception.
0x00000000004011cc in rand_fract () at random_generator.c:13
13          int value = (i * j) / (i + j);
(gdb) print i
$1 = 1
(gdb) print j
$2 = -1
(gdb) list
8      //Generate a short random number
9      short rand_fract(void) {
10         short sum = 1;
11         for (short i = 1; i < (rand() % N); ++i)
12             for (short j = 1; i < N; ++j) {
13                 int value = (i * j) / (i + j);
14                 sum += (value != 0) ? value : sum;
15             }
16         return sum;
17     }
(gdb) █
```

Figure 22 - Snapshot of debugging process

The debugger output shows that the error occurred on the same line as earlier. But in this case, the value of 'i' and 'j' are not '0,0' but they are '1, -1' which is causing the denominator at line 13 to be '0' and thus, causing an FPE. In addition to print commands, we have also issued the 'list' command which shows the nearby 10 lines of the code where the program stopped.

**You can observe that some bugs in the programs are easier to debug but some aren't.**

We will have to dig in much more to find out what is going on. Also, to be noted, we have our inner loop iterating from 1 to N (which is 100), but still the value of 'j' is printed out to be '-1'. How is this even possible!? Smart programmers would have the problem identified, but let's stick to the basics on how to gdb. Let us use the 'break' command and set a breakpoint at line number 13 and observe what is going on.

```
(gdb) list 13
8      //Generate a short random number
9      short rand_fract(void) {
10         short sum = 1;
11         for (short i = 1; i < (rand() % N); ++i)
12             for (short j = 1; j < N; ++j) {
13                 int value = (i * j) / (i + j);
14                 sum += (value != 0) ? value : sum;
15             }
16         return sum;
17     }
(gdb) break 13
Breakpoint 1 at 0x4011b5: file random_generator.c, line 13.
(gdb) info breakpoints
Num      Type           Disp Enb Address              What
1        breakpoint      keep y   0x00000000004011b5 in rand_fract at random_generator.c:13
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/.../debugger/a.out

Breakpoint 1, rand_fract () at random_generator.c:13
13                 int value = (i * j) / (i + j);
(gdb) print i
$3 = 1
(gdb) print j
$4 = 1
(gdb) █
```

Figure 23 – Setting Breakpoint

Thus, using the command 'break 13' we have set the breakpoint at line number 13 which was verified using the 'info breakpoint' command. Then, we reran the program with the 'run' command. At line 13 the program stopped and using 'print' command we checked the values of 'i' and 'j'. At this point, all seems to be well. Now, let's proceed further. For stepping 1 instruction we can use the 'step' command. Let's do that and observe the value of 'j'.



```
(gdb) print j
$5 = 1
(gdb) step
14             sum += (value != 0) ? value : sum;
(gdb) step
12             for (short j = 1; i < N; ++j) {
(gdb) step

Breakpoint 1, rand_fract () at random_generator.c:13
13             int value = (i * j) / (i + j);
(gdb) print j
$6 = 2
(gdb) step
14             sum += (value != 0) ? value : sum;
(gdb) step
12             for (short j = 1; i < N; ++j) {
(gdb) step

Breakpoint 1, rand_fract () at random_generator.c:13
13             int value = (i * j) / (i + j);
(gdb) print j
$7 = 3
(gdb) █
```

Figure 24 – single-stepping through to catch error !!

You can observe the usage of the 'step' command. We are going through the program line by line and checking the values of the variable 'j'.

There seems to be a lot of writing/typing of the 'step' command just to proceed with the program. Since we have already set a breakpoint at line 13, we can use another command called 'continue'. This command continues the program till the next breakpoint or the end of the program.

```

(gdb) continue
Continuing.

Breakpoint 1, rand_fract () at random_generator.c:13
13         int value = (i * j) / (i + j);
(gdb) print j
$8 = 4
(gdb) continue
Continuing.

Breakpoint 1, rand_fract () at random_generator.c:13
13         int value = (i * j) / (i + j);
(gdb) print j
$9 = 5
(gdb) continue
Continuing.

Breakpoint 1, rand_fract () at random_generator.c:13
13         int value = (i * j) / (i + j);
(gdb) print j
$10 = 6
(gdb) █

```

Figure 25 – Debugging continued

You can see that we reduced the typing of 'step' command by 3 times to a 'continue' command just 1 time. But this is also having us write 'continue' and 'print' multiple times.

Let us use some other utility in gdb known as 'data breakpoints' also known as watchpoints. But before that, let us delete the existing breakpoint using the 'delete' command.

```

(gdb) info breakpoints
Num      Type          Disp Enb Address                What
1        breakpoint    keep y  0x00000000004011b5 in rand_fract at random_generator.c:13
        breakpoint already hit 6 times
(gdb) delete 1
(gdb) info breakpoints
No breakpoints or watchpoints.
(gdb) █

```

Figure 26 – Debugging continued

Now let us see how to set a watchpoint.

```
(gdb) watch j
Hardware watchpoint 2: j
(gdb) info watchpoints
Num      Type          Disp Enb Address      What
2        hw watchpoint  keep y          j
(gdb) continue
Continuing.

Hardware watchpoint 2: j

Old value = 6
New value = 7
0x0000000004011f5 in rand_fract () at random_generator.c:12
12          for (short j = 1; i < N; ++j) {
(gdb)
Continuing.

Hardware watchpoint 2: j

Old value = 7
New value = 8
0x0000000004011f5 in rand_fract () at random_generator.c:12
12          for (short j = 1; i < N; ++j) {
(gdb)
Continuing.

Hardware watchpoint 2: j

Old value = 8
New value = 9
0x0000000004011f5 in rand_fract () at random_generator.c:12
12          for (short j = 1; i < N; ++j) {
(gdb) █
```

Figure 27 – Setting a watch point

Thus, using the command 'watch j' we have set a watchpoint over 'j'. Now every time when the value of 'j' changes, a break will occur. You can also note the old and new values of 'j' printed out at each break. Another point to note is that after having one 'continue' command, the program had a break. Further, by just pressing the 'Enter/Return' button on the keyboard, the continue command was repeated. Thus, by pressing the 'Enter/Return' button, the last command is repeated. At this point, we have learned much about the debugger, but we are still not able to proceed fast with our error. Is there any other way to proceed? Well, yes!!

We want to observe at the point where the value of 'j' reaches closer to 'N i.e. 100'. This means that we are only concerned about what happens after 'j' reaches 99. Here, we land upon using what is called conditional breakpoints. First, we will delete our watchpoint and then make use of the conditional breakpoint.

```
(gdb) info watchpoints
Num      Type           Disp Enb Address          What
2        hw watchpoint    keep y          j
          breakpoint already hit 4 times
(gdb) delete 2
(gdb) list 13
8          //Generate a short random number
9          short rand_fract(void) {
10         short sum = 1;
11         for (short i = 1; i < (rand() % N); ++i)
12             for (short j = 1; i < N; ++j) {
13                 int value = (i * j) / (i + j);
14                 sum += (value != 0) ? value : sum;
15             }
16         return sum;
17     }
(gdb) break random_generator.c:13 if j == 99
Note: breakpoint 3 also set at pc 0x4011b5.
Breakpoint 4 at 0x4011b5: file random_generator.c, line 13.
(gdb) continue
Continuing.

Breakpoint 3, rand_fract () at random_generator.c:13
13         int value = (i * j) / (i + j);
(gdb) print j
$12 = 99
(gdb) █
```

Figure 28 – Debugging continued

You can observe another variant of the 'break' command. We have explicitly stated the file and the line number along with a condition to stop. This is useful when the source code is large and having multiple files. After setting a conditional break, we stopped at the point where the value of 'j' becomes '99'. Now, let us see what happens next. Since this is a critical point at which we could observe the program, it is better if we step in the program using the 'step' command instead of relying on any break/watchpoints.

```

(gdb) print j
$17 = 99
(gdb) step
14             sum += (value != 0) ? value : sum;
(gdb)
12             for (short j = 1; i < N; ++j) {
(gdb)
13             int value = (i * j) / (i + j);
(gdb) print j
$18 = 100
(gdb) step
14             sum += (value != 0) ? value : sum;
(gdb)
12             for (short j = 1; i < N; ++j) {
(gdb)
13             int value = (i * j) / (i + j);
(gdb) print j
$19 = 101
(gdb) █

```

Figure 29 – Well, Back to square one !!

This is unexpected!! The value of 'j' should never be 100 or anything above it.

### Thus, something is wrong with the conditional statement!!

By observation, we have figured out that the condition is itself wrong. It should have been 'j < N' instead of 'i < N'. This is a silly mistake of the programmer that lead us to this much of an effort.

**Also, the value of 'j' which was observed as '-1' was an outcome of the 'short' datatype overflow i.e. the value of 'j' went from 1 to 32767 (assuming short as 2 bytes) and then from -32768 to -1.**

Finally, a hard programming bug was discovered. Let us correct this error and rerun the program.

```

$ gcc random_generator.c
$ ./a.out
Segmentation fault (core dumped)
$ ./a.out
1648815196934936907712847411075269363872465178968652936899126642679968327854843818024803725602089977
$ ./a.out
Segmentation fault (core dumped)
$ ./a.out
5697819555377639608368302418588269943918647330492449391532502328545856833586737093122407957112268963
$ ./a.out
6150930494475890863050318719122734582864309765193799040843958123681888230308039318234438024068348747
$ ./a.out
Segmentation fault (core dumped)
$ █

```

Figure 30 – Again Dumping Core!! Things are getting interesting or frustrating or both !!

This is strange!!

Sometimes the program is getting the correct output, but sometimes, we are getting a segmentation fault. Debugging such a program may be tricky since the occurrence of the bug is low. We will proceed with our standard debugger steps to identify the error.

```
$ gcc -g random_generator.c
$ gdb a.out
GNU gdb (GDB) Fedora 8.3.50.20190824-25.fc31
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) set style enabled off
(gdb) run
Starting program: /home/.../debugger/a.out
5411371059776263605409873043180521681086975694174815924540859823191291008689026600122878853935366497
[Inferior 1 (process 61832) exited normally]
(gdb)
```

Figure 31 – Debugging continued

We compiled the code and ran it using the debugger. But the program was completed successfully. Let us rerun it till the point where the program fails.

```
(gdb) run
Starting program: /home/.../debugger/a.out
5411371059776263605409873043180521681086975694174815924540859823191291008689026600122878853935366497
[Inferior 1 (process 61832) exited normally]
(gdb) run
Starting program: /home/.../debugger/a.out
7919846386128432134671007571802513619267000845358948048917009272836772572766214308134147179016591178
[Inferior 1 (process 61978) exited normally]
(gdb) run
Starting program: /home/.../debugger/a.out

Program received signal SIGSEGV, Segmentation fault.
0x000000000040126c in factorial (x=4294792703) at random_generator.c:24
24      return (x * factorial(x - 1));
(gdb)
```

Figure 32 – Debugging continued

Here we observe a point where the program exited at the function 'factorial'.

This is a point where the debugger didn't give much information about what the value of the variable 'x' was. It just pointed out that the program failed at the function named 'factorial'. That's it!

Another reason for such kind of output would be the recursive nature of the function. The stack frame where the function 'factorial' failed could be in a long nest of recursive calls. At such points, it would be better to inspect the program at an earlier point

and look for errors. Let us have a breakpoint before the 'factorial' function was called and view the value of the parameters that are passed to the function.

```
(gdb) list main
22         return 1LL;
23     else
24         return (x * factorial(x - 1));
25     }
26
27     int main (int argc, char *argv[]) {
28
29         short f1 = 0;
30
31         //Create a random seed based on process id.
(gdb)
32         srand((unsigned int) getpid());
33
34         //Generate a random number salt.
35         f1 = rand_fract() % 10;
36
37         //Get the factorial of the number
38         long long random_fact = factorial(f1);
39
40         //Normalize the factorial to number modulo N_LEN + 1
41         int normalized_fact = random_fact % (N_LEN + 1);
(gdb) break 36
Breakpoint 1 at 0x4012da: file random_generator.c, line 38.
(gdb) run
Starting program: /home/vineetm/debugger/a.out

Breakpoint 1, main (argc=1, argv=0x7fffffff0d8) at random_generator.c:38
38         long long random_fact = factorial(f1);
(gdb) print f1
$1 = 1
(gdb) continue
Continuing.
9962554943440906583333593426000827274699155147995250801174774876796185292736525250533642241728519329
[Inferior 1 (process 62328) exited normally]
(gdb) █
```

Figure 33 – Debugging continued (Will it ever end?)

Thus, we have set a breakpoint before the call of the function 'factorial' and ran the program. For the value of 'f1 = 8' for the 'factorial' function, the process seems to exit normally. Let us rerun.

```
(gdb) run
Starting program: /home/██████████/debugger/a.out

Breakpoint 1, main (argc=1, argv=0x7fffffff0d8) at random_generator.c:38
38         long long random_fact = factorial(f1);
(gdb) print f1
$1 = -8
(gdb) continue
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x00000000040126c in factorial (x=4294792699) at random_generator.c:24
24         return (x * factorial(x - 1));
(gdb) █
```

Figure 34 – We are almost there !!

Unexpectedly, we have got the value of 'f1' as '-8' and the program seems to have crashed. Let us observe the 'rand\_fract' function and 'factorial' function once again. And study the behavior of the functions where we could get a negative number.

```
(gdb) list rand_fract
4
5     #define N 100
6     #define N_LEN 100
7
8     //Generate a short random number
9     short rand_fract(void) {
10        short sum = 1;
11        for (short i = 1; i < (rand() % N); ++i)
12            for (short j = 1; j < N; ++j) {
13                int value = (i * j) / (i + j);
(gdb)
14                sum += (value != 0) ? value : sum;
15            }
16        return sum;
17    }
18
19    //Returns the factorial of a number
20    long long factorial(unsigned int x) {
21        if (x == 1 || x == 0)
22            return 1LL;
23        else
(gdb) run
Starting program: /home/██████████/debugger/a.out

Breakpoint 1, main (argc=1, argv=0x7fffffff0d8) at random_generator.c:38
38        long long random_fact = factorial(f1);
(gdb) print f1
$2 = -8
(gdb) █
```

Figure 35 – Debugging continued

Important points here to observe are:

The 'rand\_fract' function is returning a datatype of 'short' while the calculation of the return value could be significantly large which may overflow the size of 'short', thus, causing a negative answer.

The function 'factorial' is expecting a value of type 'unsigned int'. Since the value passed to the function is a negative value, having an implicit conversion from a negative number to an unsigned number means that we are having a very large value passed to the factorial function.

Also, since the 'factorial' function is recursive, passing a very large number to it could cause multiple calls to the same function and thus, overflowing the stack provided to the user.



Now let us, step further into our program and see whether what we are discussing is the same behavior that is being observed.

```
(gdb) print f1
$4 = -8
(gdb) step
factorial (x=4294967288) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb)
24         return (x * factorial(x - 1));
(gdb)
factorial (x=4294967287) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb)
24         return (x * factorial(x - 1));
(gdb)
factorial (x=4294967286) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb)
24         return (x * factorial(x - 1));
(gdb)
factorial (x=4294967285) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb)
24         return (x * factorial(x - 1));
(gdb)
factorial (x=4294967284) at random_generator.c:21
21         if (x == 1 || x == 0)
(gdb) █
```

Figure 36 – At last a clue!!!

This is what we had expected!!

**A number '-1' passed to the 'factorial' function is being implicitly converted to a very large number '4294967295'.**

Stepping in more reveals the recursive behavior of the 'factorial' function i.e. each call is having a sub-call to the same function with one value less. Thus, what to do in these types of cases. Assume you have a large code where these functions are called from multiple locations.

Modifying the signature of any of the functions means changing the code everywhere where the function is called. This is not affordable!! These are some cases, where a choice is to be made where patching the code is necessary for the semantics of the program.

Let us observe a piece of code where this change can be made and then test our program for the expected results.

```
int main (int argc, char *argv[]) {  
  
    short f1 = 0;  
  
    //Create a random seed based on process id.  
    srand((unsigned int) getpid());  
  
    //Generate a random number salt.  
    f1 = rand_fract() % 10;  
  
    f1 = abs(f1);  
  
    //Get the factorial of the number  
    long long random_fact = factorial(f1);  
  
    //Normalize the factorial to number modulo N_LEN + 1  
    int normalized_fact = random_fact % (N_LEN + 1);  
  
    int *array = NULL;
```

Figure 37 - Correction applied !!

By observing the code, we find out that the expected value of 'f1' is between '0 to 9' (because of the modulo 10 operation).

Thus, without changing the signature of any function, we have inserted a patch (the highlighted) portion, that maintains the semantics of the code as well cures the problem that we had. Now let us just run and check our final program.

```
$ gcc random_generator.c
$ ./a.out
1947155904444356260827867895829013940560127574392384362061544757042318542200659899527743928595211645
$ ./a.out
0929989745546167856100961512939018573760223504833542534886091294243732854126729096261941760801537820
$ ./a.out
0244202592758390536991444038465396583053516022410228562188134665524049393105566500577005828487059653
$ ./a.out
2872718293228567054539368096969066437379893671576029177909795701346393295764931536773483363035181911
$ ./a.out
9128766061538956027759598074797832715087451437704122190965898083361413690723150214543517739636518290
$ ./a.out
4700580792312412551673394453147630608790931492649027378923259025287077290331618510470262819931652479
$ ./a.out
3597977632870365479023130705918446909083470263354375991983675631252252710058384841530848408963208645
$ ./a.out
0864510419056291282368845079139095210792697191764209304803037158672651132052448868790301906812889064
$ ./a.out
4972916609538445900529958158240849030612776510222275380497441425328380877450674923651890544608240290
$ ./a.out
9528608642866177753983842182285047120984190000785095691019238964676666205506776407087180325311790389
$ █
```

Figure 38 – Resolved !!!

Thus, we are getting the correct results as expected.

## Conclusions

We started with a program that we assumed to be functional but then the program ended up with bugs that were not straightforward. We then explored the power of the debugger and the various ways to identify the bugs in our program. We looked upon the easy solutions and slowly migrated towards the type of bugs that are not easily traceable.

Finally, we identified and corrected all the bugs in our program with the help of the debugger and arrived at a bug-free code.

## Points to Note

- Bugs in the program cannot be necessarily a compilation error.
- One type of error can be caused by multiple bugs in the same line of code.
- Sometimes, it is not possible to change the code even when the problem is identified. The best way to cure this is to study the behavior of the code and apply patches wherever necessary.
- Using simple utilities from the 'GNU Debugger' can help in getting rid of problem-causing bugs in large programs.

## Overall Coding Modifications Done

```

random_generator.c
//Generate a short random number
short rand_fract(void) {
    short sum = 1;
    for (short i = 1; i < (rand() % N); ++i)
        for (short j = 1; j < N; ++j) {
            int value = (i * j) / (i + j);
            sum += (value != 0) ? value : sum;
        }
    return sum;
}

//Returns the factorial of a number
long long factorial(unsigned int x) {
    if (x == 1 || x == 0)
        return 1LL;
    else
        return (x * factorial(x - 1));
}

int main (int argc, char *argv[]) {
    short f1 = 0;

    //Create a random seed based on process id.
    srand((unsigned int) getpid());

    //Generate a random number salt.
    f1 = rand_fract() % 10;

    f1 = abs(f1);

    //Get the factorial of the number
    long long random_fact = factorial(f1);
}

random_generator_buggy.c
//Generate a short random number
short rand_fract(void) {
    short sum = 1;
    for (short i = 0; i < (rand() % N); ++i)
        for (short j = 0; j < N; ++j) {
            int value = (i * j) / (i + j);
            sum += (value != 0) ? value : sum;
        }
    return sum;
}

//Returns the factorial of a number
long long factorial(unsigned int x) {
    if (x == 1 || x == 0)
        return 1LL;
    else
        return (x * factorial(x - 1));
}

int main (int argc, char *argv[]) {
    short f1 = 0;

    //Create a random seed based on process id.
    srand((unsigned int) getpid());

    //Generate a random number salt.
    f1 = rand_fract() % 10;

    //Get the factorial of the number
    long long random_fact = factorial(f1);

    //Normalize the factorial to number modulo N LEN +
}

```

Unicode (UTF-8) C Ln 68, Col 2

Unicode (UTF-8) C Ln 11, Col 1

Figure 39 – What all we did to get things right!

# Machine Learning / Deep Learning Application Development

Most of the popular python based machine learning/deep learning libraries are installed on the PARAM Yukti system. While developing and testing their applications, users have the option to choose different environment/runtime setup like “virtual environment-based python libraries” or “conda runtime based python libraries”.

For most of the major environments (virenv, conda) different modules are prepared. Users can check the list of the modules by using “**module avail**” command. Shown below is an example of loading conda environment in the current bash shell and continue with application development.

Once logged into PARAM Yukti HPC Cluster, check which all libraries are available, loaded in the current shell. To checklist of modules loaded in the current shell, use the command given below:

```
$ module list
```

To check all modules available on the system, but not loaded currently, use the command given below:

```
$ module avail
```

To activate conda environment on PARAM Yukti, load module “conda-python/3.7” as shown below:

```
$ module load conda-python/3.7
```

Conda environment has been installed with most of the popular python packages as shown below

Tensorflow	Tensorflow-gpu	Mpi4py	Keras
Theano	Scipy	Scikit-Learn	Pytorch

Once “conda-python/3.7” module is loaded, end-users can use all libraries inside their python program. Many other modules based on virtual env are available on the system.

Users can load those libraries using “module load” command and use them for their applications.

## How to Install your own Software?

There are two approaches to install the software.

1. System-wide installation
2. Local installation.

System-wide installation can be done by only the admin. If you wish to do this, please approach the system administrator. Users can do the local installation in their home directory. In this section, we are describing the installation of HMMER application in the user's home directory.

### Local installation

**Step 1.** Login to the Yukti cluster by using your credential.

**Step 2.** Download the software that you want to install. For example to download HMMER software use the command given below.

```
$ wget http://eddylib.org/software/hmmer/hmmer.tar.gz
```

**Step 3.** Untar the file. ( if your software is in zip format use unzip command)

```
$ tar zxf hmmer.tar.gz
```

**Step 4.** go to the software folder.

```
$ cd hmmer-3.3
```

**Step 5.** configure the installation path.

```
$ ./configure --prefix /your/install/path
```

**Step 6.** now run the 'make' command to install the software on the installation path.

```
$ make
```

The newly compiled binaries are now in the src directory.

**Step 7.** Runs a test suite that checks for errors in the software (optional)

```
$ make check
```

**Step 8.** run 'make install' to install the programs and man pages in your location mention in step 2 #

```
$ make install
```

By default, programs are installed in **/usr/local/bin** and man pages in **/usr/local/share/man/man1/**, if you do not provide installation path in step 2.

\* This is general instruction for installation, please refer to the installation instruction or manual or readme file that comes with software for more details.

# if you get any dependency error, resolve that or ask the system admin to install that dependency if not installed.

Reference link: <http://hmmer.org/documentation.html>

# Some Important Facts

## About File Size

The global/home is served by a number of storage arrays. Each storage array contains a portion of the global/home. The size of a disk in the storage array is 2TB (2000 GB).

Technically, the size of a file can be about 2000 GB (which is really big). However, since the disk is shared by a large number of files, effectively the size of a single file will be far smaller. Normally, this file size is kept to be about a few GBs which is sufficient for most of the users. However, if you wish to have file sizes that are larger than this, you need to create files

ACROSS disks and this process is known as 'striping'.

```
lfs setstripe -c 4 .
```

After this has been done all new files created in the current directory will be spread over 4 storage arrays each having 1/4th of the file. The file can be accessed as normal no special action needs to be taken. When the striping is set this way, it will be defined on a per-directory basis so different directories can have different stripe setups in the same file system, new subdirectories will inherit the striping from its parent at the time of creation.

We recommend users set the stripe count so that each chunk will be approx. 200-300GB each, for example

File Size	Stripe count	Command
500-1000 GB	4	lfs setstripe -c 4 .
1000 – 2000 GB	8	lfs setstripe -c 8

Once a file is created with a stripe count, it cannot be changed. User by themselves are also able to set stripe size and stripe count for their directories and A user can check the set stripe size and stripe count with the command:

```
lfs getstripe <path to the direcorey>
```

To set the stripe count as

```
lfs setstripe -c 4 -s 10m <path to the direcorey>
```



The options on the above command used have these respective functions.

- **-c** to set the stripe count; 0 means use the system default (usually 1) and -1 means stripe overall available OSTs (lustre Object Storage Targets).
- **-s** to set the stripe size; 0 means use the system default (usually 1 MB) otherwise use k, m, or g for KB, MB, or GB respectively

## Little-Endian and Big-Endian issues?

By and large, most of the computers follow little-endian format. This essentially means that the last byte of the binary representation of data is stored first. However, there is another way of representing data (used in some machines) where the first byte of the binary representation of data is stored first. When binary files are to be read across these different kinds of machines, bytes need to be re-ordered. Many compilers do support this feature.

Please explore this aspect, if a perfectly working code on a given machine, fails to get executed by another machine (with a different processor).

## Best Practices for HPC

---

1. Do **NOT** run any job which is long a few minutes on the login nodes. The login node is for the compilation of jobs. It is best to run the job on the compute nodes.
2. It is **recommended** to go through the beginner's guide in **/home/apps/Docs/samples**. This should serve as a good starting point for the new users.
3. Use the same compiler to compile different parts/modules/library-dependencies of an application. Using different compilers (e.g. pgcc + icc) to compile different parts of the application may cause linking or execution issues.
4. Choosing appropriate compiler switches/flags/options (e.g. -O3) may increase the performance of the application substantially (accuracy of output must be verified). Please refer to the documentation of compilers (online / docs present inside compiler installation path/man pages etc.)
5. Modules/libraries used for execution should be the same as those used for compilations. This can be specified in the job submission script.
6. Be aware of the amount of disk space utilized by your job(s). Do an estimate before submitting multiple jobs.
7. Please submit jobs preferably in \$SCRATCH. You can back up your results/summaries in your \$HOME
8. \$SCRATCH is NOT backed up! Please download all your data to your Desktop/ Laptop.
9. Before installing any software in your home, ensure that it is from a reliable and safe source. Ransomware is on the rise!
10. Please do not use spaces while creating the directories and files.
11. Please inform PARAM Yukti support when you notice something strange - e.g. unexpected slowdowns, files missing/corrupted etc.

## Installed Applications/Libraries

Following is the list of a few of the applications from various domains of science and engineering installed in the system.

<b>HPC Applications</b>	Bio-informatics	MUMmer, HMMER, MEME, Schrodinger, PHYLIP, mpiBLAST, ClustalW,
	Molecular Dynamics	NAMD (for CPU and GPU), LAMMPS, GROMACS
	Material Modeling, Quantum Chemistry	Quantum-Espresso, Abinit, CP2K, NWChem,
	CFD	OpenFOAM, SU2
	Weather, Ocean, Climate	WRF-ARW, WPS (WRF), ARWPost (WRF), RegCM, MOM, ROMS
<b>Deep Learning Libraries</b>	cuDNN, TensorFlow, Tensorflow with Intel Python , Tensorflow with GPU, Theano, Caffe , Keras , numpy, Scipy, Scikit-Learn, pytorch.	
<b>Visualization Programs</b>	GrADS, ParaView, VisIt, VMD	
<b>Dependency Libraries</b>	NetCDF, PNETCDF, Jasper, HDF5, Tcl, Boost, FFTW	

### Standard Application Programs on PARAM Yukti

The purpose of this section is to expose the users to different application packages which have been installed. Users interested in exploring these packages may kindly go through the scripts, typical input files, and typical output files. It is suggested that, at first, the users may submit the scripts provided and get a feel of executing the codes. Later, they may change the parameters and the script to meet their application requirements.

## LAMMPS Applications

**LAMMPS** is an acronym for **L**arge-scale **A**tomic/ **M**olecular **M**assively **P**arallel **S**imulator. This is extensively used in the fields of Material Science, Physics, Chemistry, and many others.

More information about LAMMPS may please be found at <https://lammps.sandia.gov> .

1. The LAMMPS input is **in.lj** file which contains the below parameters.

**Input file = in.lj**

```
# 3d Lennard-Jones
melt
variable index 1
variable index 1
variable xx equal 64*$x
variable yy equal 64*$y
variable zz equal 64*$z
units lj
atom_style atom
lattice fcc 0.8442
region box block 0 ${xx} 0 ${yy} 0
create_box 1 box
create_atoms 1
velocity all create 1.44 87287 loop
pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0
neighbor 0.3 bin
neighbor_modify delay 0 every 20 check
fix 1 all
k 1000
u 000
```

2. THE LAMMPS RUNNING SCRIPT

```
#!/bin/sh
#SBATCH -N 8
#SBATCH
--ntasks-per-node=40
#SBATCH --time=08:50:20
#SBATCH --job-name=lammps
```

```

#SBATCH
--error=job.%J.err_8_node_40
#SBATCH
--output=job.%J.out_8_node_40
#SBATCH --partition=standard

module load compiler/intel/2018.2.199
module load

compiler/intel-mpi/mpi-2018.2.199 module

load compiler/gcc/7.3.0

source
/opt/ohpc/pub/apps/intel/2018_2/compilers_and_libraries_2018.2.1
99/1 linux/mkl/bin/mklvars.sh intel64

export
I_MPI_FALLBACK=disable
export
I_MPI_FABRICS=shm:ofa
#export
I_MPI_FABRICS=shm:tmi
#export
I_MPI_FABRICS=shm:dapl
export I_MPI_DEBUG=5

cd

```

```

/home/manjunath/NEW_LAMMPS/lammps-7Aug19/bench

```

```

export OMP_NUM_THREADS=1

```

```

time mpiexec.hydra -n $SLURM_NTASKS -genv OMP_NUM_THREADS 1
/home/manjunath/NEW_LAMMPS/lammps-7Aug19/src/lmp intel cpu intelm

```

### 3. LAMMPS OUTPUT FILE.

```

LAMMPS (7 Aug 2019)
  using 1 OpenMP thread(s) per MPI task
Lattice spacing in x,y,z = 1.6796 1.6796
1.6796
Created orthogonal box = (0 0 0) to (107.494 107.494 107.494)
  5 by 8 by 8 MPI processor grid
Created 1048576 atoms
  create_atoms CPU = 0.00387692 secs
Neighbor list info ...
  update every 20 steps, delay 0 steps, check
  no max neighbors/atom: 2000, page size:
  100000 master list distance cutoff = 2.8
  ghost atom cutoff = 2.8
  binsize = 1.4, bins = 77 77
  77
  1 neighbor lists, perpetual/occasional/extra = 1 0 0
  (1) pair lj/cut, perpetual
      attributes: half, newton
      on
      pair build:
      half/bin/atomonly/newton stencil:
      half/bin/3d/newton.....
      bin: standard
Setting up Verlet run
...
Unit style      : lj
Current step    : 0
Time step       : 0.005

```

```

Per MPI rank memory allocation (min/avg/max) = 3.154 | 3.156 |
3.162 Mbytes
Step Temp E_pair E_mol TotEng Press
      0          1.44   -6.7733681          0   -4.6133701   -
5.0196704
1000000  0.65684946   -5.7123998          0   -4.7271266
0.49078272
Loop time of 2955.97 on 320 procs for 1000000 steps with
1048576 atoms

Performance: 146145.063 tau/day, 338.299
timesteps/s 99.4% CPU use with 320 MPI tasks x 1
OpenMP threads

MPI task timing breakdown:
Section | min time | avg time | max time | %varavg | %total
Pair    | 1284.2   | 1512.3   | 1866.9   | 494.3   | 51.16
Neigh   | 178.94   | 207.58   | 261.09   | 217.8   | 7.02
Comm    | 793.59   | 1207.7   | 1468.3   | 654.3   | 40.86
Output  | 0.00011516 | 0.00084956 | 0.0027411 | 0.0   | 0.00
Modify  | 19.566   | 22.639   | 29.863   | 67.3   | 0.77
Other   |          | 5.744    |          |         | 0.19

Nlocal:   3276.8 ave 3325 max 3231 min
Histogram: 4 7 21 63 67 80 50 22 5 1
Nghost:   5011.29 ave 5063 max 4956 min
Histogram: 5 9 26 45 57 76 51 34 12 5
Neighs:   122781 ave 127005 max 118605 min
Histogram: 3 5 36 59 63 52 66 24 11 1

Total # of neighbors =
39290074 Ave neighs/atom =
37.4699 Neighbor list builds =
50000 Dangerous builds not
checked Total wall time:
0:49:15

```

## GROMACS APPLICATION

### GROMACS

GROningen MACHine for Chemical Simulations (GROMACS) is a [molecular dynamics](#) package mainly designed for simulations of [proteins](#), [lipids](#), and [nucleic acids](#). It was originally developed in the Biophysical Chemistry department of [University of Groningen](#), and is now maintained by contributors in universities and research centers worldwide. GROMACS is one of the fastest and most popular software packages available and can run on [central processing units](#) (CPUs) and [graphics processing units](#) (GPUs).

## Input description of Gromacs

The input file can be download from

[ftp://ftp.gromacs.org/pub/benchmarks/water\\_GMX50\\_bare.tar.gz](ftp://ftp.gromacs.org/pub/benchmarks/water_GMX50_bare.tar.gz)

The mdp option used is pme with 50000 steps

### Submission Script:

```
#!/bin/sh
#SBATCH -N
10
#SBATCH
--ntasks-per-node=48
##SBATCH --time=03:05:30
#SBATCH --job-name=gromacs
#SBATCH --error=job.16.%J.err
#SBATCH
--output=job.16.%J.out
#SBATCH --partition=standard

cd
/home/shweta/water-cut1.0_GMX50_bare/3072
module load compiler/intel/2018.5.274
module load apps/gromacs/5.1.4/cpu/intel_18.5

export I_MPI_DEBUG=5
export
OMP_NUM_THREADS=1
mpirun -np 4 gmx_mpi grompp -f pme.mdp -c conf.gro -p topol.top

time mpirun -np $SLURM_NTASKS gmx_mpi mdrun -s topol.tpr) 2>&1 |
tee log_gromacs_40_50k_mpirun
```

### Output Snippet:

Number of logical cores detected (48) does not match the number reported by OpenMP (1).

Consider setting the launch configuration manually!

Running on 10 nodes with total 192 cores, 480 logical cores

Cores per node: 0 - 48

Logical cores per node: 48

Hardware detected on host cn072 (the node of MPI rank 0):

CPU info:

Vendor: GenuineIntel

Brand: Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz

SIMD instructions most likely to fit this hardware: AVX2\_256

SIMD instructions selected at GROMACS compile time: AVX2\_256

Reading file /home/shweta/Gromacs/water-cut1.0\_GMX50\_bare/3072/topol.tpr,

VERSION 5.1.4 (single precision)

Changing nstlist from 10 to 20, rlist from 1 to 1.032

The number of OpenMP threads was set by environment variable

OMP\_NUM\_THREADS to 1 (and the command-line setting agreed with that)

NOTE: KMP\_AFFINITY set, will turn off gmx mdrun internal affinity setting as the two can conflict and cause performance degradation. To keep using the gmx mdrun internal affinity setting, set the KMP\_AFFINITY=disabled environment variable.

Overriding nsteps with value passed on the command line: 50000 steps, 100 ps

Will use 360 particle-particle and 120 PME only ranks

This is a guess, check the performance at the end of the log file

Using 480 MPI processes

Using 1 OpenMP thread per MPI process

Back Off! I just backed up ener.edr to ./#ener.edr.2#



starting mdrun 'Water'  
 50000 steps, 100.0 ps.

Average load imbalance: 5.5 %  
 Part of the total run time spent waiting due to load imbalance: 3.0  
 % Average PME mesh/force load: 1.252  
 Part of the total run time spent waiting due to PP/PME imbalance: 13.2  
 %

NOTE: 13.2 % performance was lost because the PME ranks had more work to do than the PP ranks. You might want to increase the number of PME ranks or increase the cut-off and the grid

	Core t	Wall t	(
	(s) Time:	(s)	%)
	204872.624	427.	47884
Performanc	(ns/d	847	.5
e:	ay)	(hour/n	
	20.	s)	
	195	1.188	

# Acknowledging the National Supercomputing Mission in Publications

If you use supercomputers and services provided under the National Supercomputing Mission, Government of India, please let us know of any published results including Student Thesis, Conference Papers, Journal Papers, and patents obtained.

Please acknowledge the National Supercomputing Mission as given below:

The support and the resources provided by PARAM Yukti Facility under the National Supercomputing Mission, Government of India at the Jawaharlal Nehru Centre for Advanced Scientific Research (JNCASR); Bangalore are gratefully acknowledged.

Also, please submit the copies of dissertations, reports, reprints, and URLs in which “National Supercomputing Mission, Government of India” is acknowledged to:

HoD HPC Technologies,  
Centre for Development of Advanced  
Computing, CDAC Innovation Park,  
S.N. 34/B/1,  
Panchavati,  
Pashan, Pune –  
411008  
Maharashtra

Communication of your achievements using resources provided by the National Supercomputing Mission will help the Mission in measuring outcomes and gauging future requirements. This will also help in further augmentation of resources at a given site of the National Supercomputing Mission.

## Getting Help – PARAM Yukti Support

---

We suggest that you please refer to these four easy steps to generate a Ticket related to the issue you are experiencing.

Your Ticket will be assisted by the Yukti Support team. The ticket generated will be closed only when the related issue gets resolved.

You can generate a new ticket for any of the new issues that you are experiencing.

### Steps to Create a New Ticket

1. Place the URL (<https://paramyukti.jncasr.ac.in/support>) in your browser.
2. On the right-top corner of the page click **Sign In**. Refer to Fig: 36 for the same.

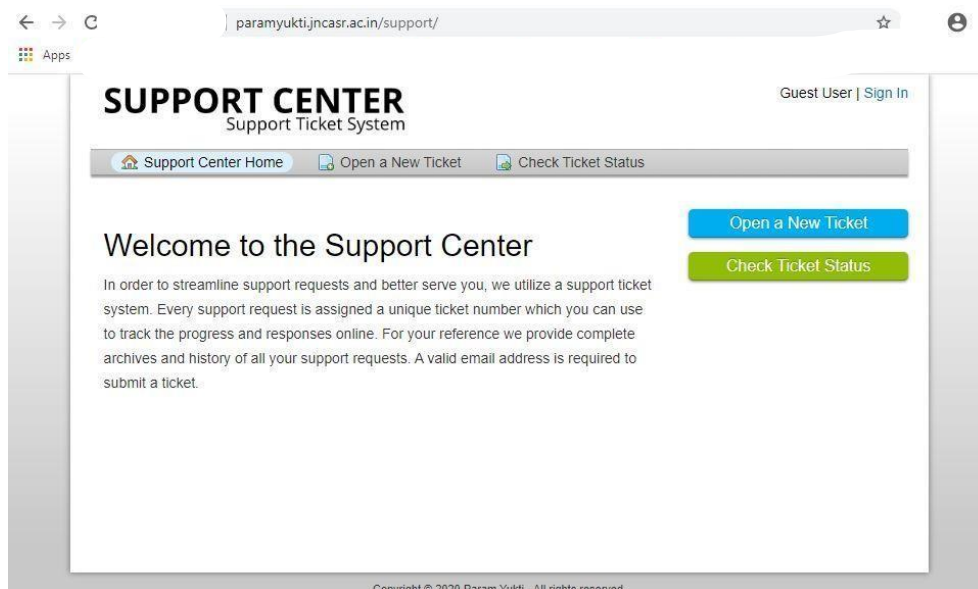


Figure 40 – Snapshot of Ticketing System

3. Sign in by using the Username and Password that you use for logging to the Cluster. Refer to Fig37 for the same.

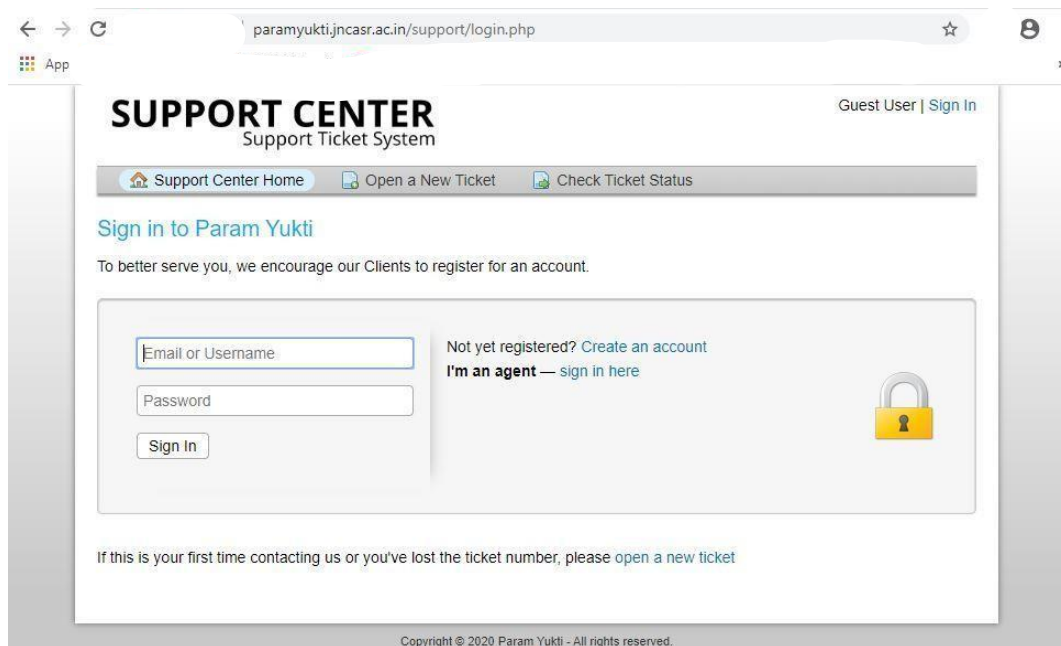


Figure 41- Snapshot of Ticketing System

4. Select a **Help Topic** from the Dropdown and then Click on **Create Ticket**. Refer to Fig:38 for the same

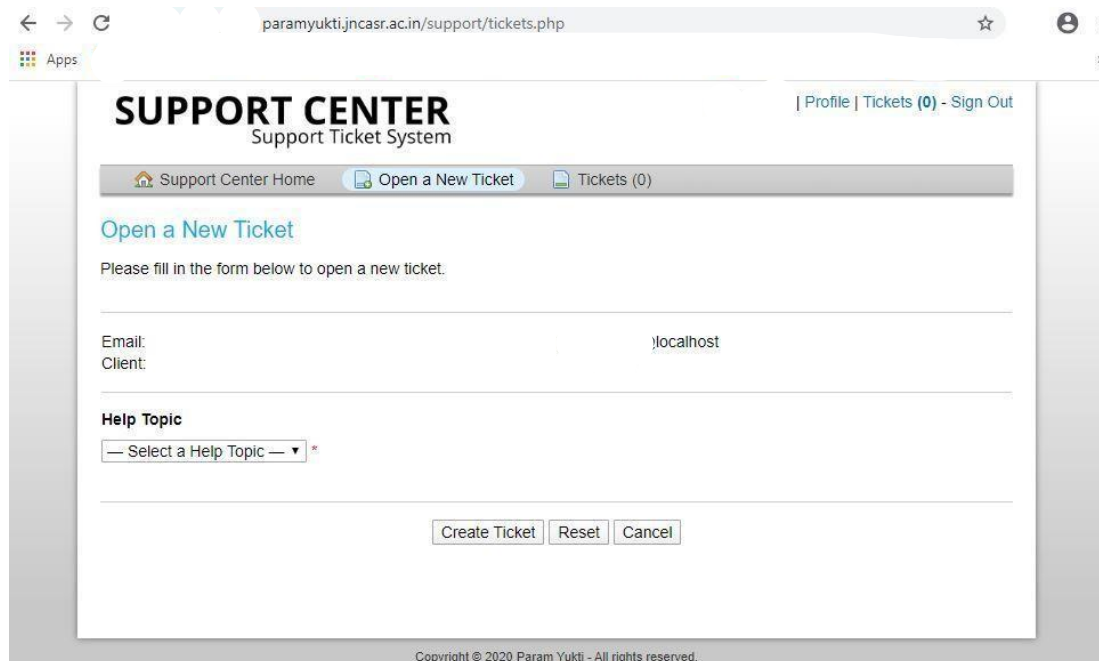


Figure 42 - Snapshot of Ticketing System

5. Please fill in the details of your issue in the fields given and then click on Create ticket.

The screenshot shows a web browser window with the URL `paramyukti.jncasr.ac.in/support/tickets.php`. The page title is "SUPPORT CENTER Support Ticket System". The navigation bar includes "Support Center Home", "Open a New Ticket", and "Tickets (0)". The main content area is titled "Open a New Ticket" and contains a form with the following fields:

- Email:** A text input field.
- Client:** A text input field.
- Help Topic:** A dropdown menu with "System Support" selected.
- Ticket Details:** A section titled "Please Describe Your Issue" containing:
  - Issue Summary:** A text input field.
  - Rich Text Editor:** A text area with a toolbar for text formatting (bold, italic, underline, link, unlink, list, etc.) and a placeholder text "Details on the reason(s) for opening the ticket".
  - File Upload:** A dashed box with the text "Drop files here or choose them".

At the bottom of the form are three buttons: "Create Ticket", "Reset", and "Cancel". The footer of the page reads "Copyright © 2020 Param Yukti - All rights reserved."

Figure 43 - Snapshot of Ticketing System

Once the Ticket is generated, an acknowledgment e-mail will be sent to your official e-mail address. The e-mail will also contain the Ticket number along with reference to the ticket that you have generated.

In case of any difficulty while accessing Yukti Support you can reach us via e-mail at [yuktisupport@jncasr.ac.in](mailto:yuktisupport@jncasr.ac.in)

## Closing Your Account on PARAM Yukti

When once you have completed your research work and you no longer need to use PARAM Yukti, you may please close your account on PARAM Yukti. Please raise a ticket by following the URL <https://paramyukti.jncasr.ac.in/support> The system administrator will guide you about the “Closure Procedure”. You will need clearance from your project coordinator/ Supervisor/ Head of the Department about you having surrendered this resource for getting a “no dues” certificate from the institute.

# References

---

1. <https://lammps.sandia.gov/>
2. <https://www.openacc.org/>
3. <https://www.openmp.org/>
4. <https://computing.llnl.gov/tutorials/mpi/>
5. <https://developer.nvidia.com/cuda-zone>
6. <https://www.mmm.ucar.edu/weather-research-and-forecasting-model>
7. <http://www.gromacs.org/>
8. <https://www.openfoam.com/>
9. <https://slurm.schedmd.com/>
10. [https://www.tutorialspoint.com/gnu\\_debugger/what\\_is\\_gdb.htm](https://www.tutorialspoint.com/gnu_debugger/what_is_gdb.htm)
11. <https://nsmindia.in/>
12. [https://en.wikipedia.org/wiki/Deep\\_learning](https://en.wikipedia.org/wiki/Deep_learning)
13. <https://docs.conda.io/en/latest/>
14. <https://docs.conda.io/en/latest/miniconda.html>
15. <https://www.tensorflow.org/>
16. <https://www.tensorflow.org/install>
17. <https://github.com/PaddlePaddle/Paddle>
18. Keras, <https://keras.io/>
19. Pytorch, <https://pytorch.org>
20. <https://mxnet.apache.org>
21. <https://software.intel.com/en-us/distribution-for-python>
22. <https://software.intel.com/en-us/articles/intel-optimization-for-tensorflow-installation-guide>