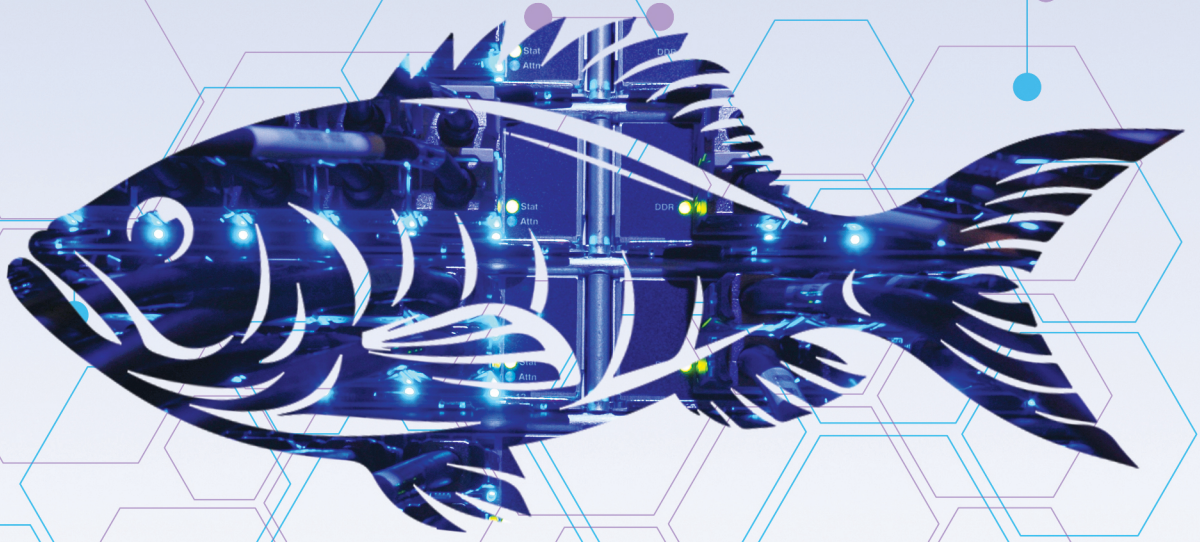
CMFRI Training Manual Series No.33/2024

# Working Manual:

# High Performance Computing Facility

# (Fish@CMFRI)



ICAR-Central Marine Fisheries Research Institute
(Department of Agricultural Research and Education, Government of India)
P.B. No. 1603, Ernakulam North P.O., Kochi - 682 018, Kerala, India


**Working Manual: High Performance Computing Facility (Fish@CMFRI)**
CMFRI Training Manual Series No.33/2024

Published by:
Dr. A. Gopalakrishnan
Director
ICAR-Central Marine Fisheries Research Institute (ICAR-CMFRI)
P.B.NO.1603, Ernakulum North P.O., Kochi- 682018, Kerala, India
www.cmfri.org.in, Email: director.cmfri@icar.gov.in
Tel. No. +91-484-2394867; Fax. No. +91-484-2394909

Compiled & Edited by:
Eldho Varghese, J Jayasankar, Manu V K, Grinson George and Thejus T V

Cover Design: Abhilash P R

Publication, Production & Co-ordination:
Arun Surendran, Library & Documentation Centre, CMFRI



Suggested Citation: Eldho Varghese, J Jayasankar, Manu V K, Grinson George and Thejus T V (2024) Working Manual: High Performance Computing Facility (Fish@CMFRI), CMFRI Training Manual Series No.33/2024, ICAR-Central Marine Fisheries Research Institute, Kochi, 50p.

CMFRI Training Manual Series No.33/2024

# Working Manual:
# High Performance Computing Facility
# (Fish@CMFRI)

**ICAR-Central Marine Fisheries Research Institute**
(Department of Agricultural Research and Education, Government of India)
P.B. No. 1603, Ernakulam North P.O., Kochi - 682 018, Kerala, India

# Preface

The advent of high-performance computing has ushered in a new era of scientific exploration, enabling researchers to address complex and data-intensive challenges with remarkable speed, accuracy, and efficiency. This manual is meticulously crafted to serve as a comprehensive guide for scientists, research scholars, and students who will harness the cutting-edge capabilities of the High-Performance Computational Facility (FISH@CMFRI). Strategically installed at the Agricultural Knowledge Management Unit (AKMU) of ICAR-CMFRI, Kochi, this state-of-the-art facility epitomizes the integration of advanced computational infrastructure into the scientific research ecosystem.

Developed with partial financial support under the prestigious DST-REVIVAL project *REhabilitation of Vibrio Infested waters of VembanAd Lake: Pollution and Solution*, the FISH@CMFRI facility represents a transformative leap forward in computational power and versatility. It not only underscores a commitment to addressing pressing environmental challenges but also exemplifies a broader vision to elevate the capabilities of the academic and research community, fostering excellence in computational science and innovation.

This facility is poised to catalyse research endeavours, significantly benefiting the academic and scientific community. Furthermore, it plays a pivotal role in capacity building by fostering a workforce skilled in high-performance computing techniques. By integrating cutting-edge emerging technologies, this facility also promotes innovation in research and teaching, laying the foundation for transformative scientific advancements.

We hope this manual serves as a valuable resource for all users, enhancing their experience with the facility and supporting their pursuit of excellence in research.

# Table of Contents

# Introduction

High-speed computing has been the "go to" option for many institutions doing research and development in biological science, especially natural resources. With the platform that usually hosts such research initiatives being multipronged and multi-sourced, data collation and analysis gets even more computer intensive. In any science-based initiative, it is next to impossible to have a data template which is homogenous, equispaced and equidistributed. More so for any other initiative that could be planned in marine fisheries scenarios. The inputs could be of any kind from organized tabulated data to sudden pieces of messages leading to actionable information akin to the Internet of Things and more rigorous systematic input from concerted exercises lie downscaling from macro-regional models like the ROMs.

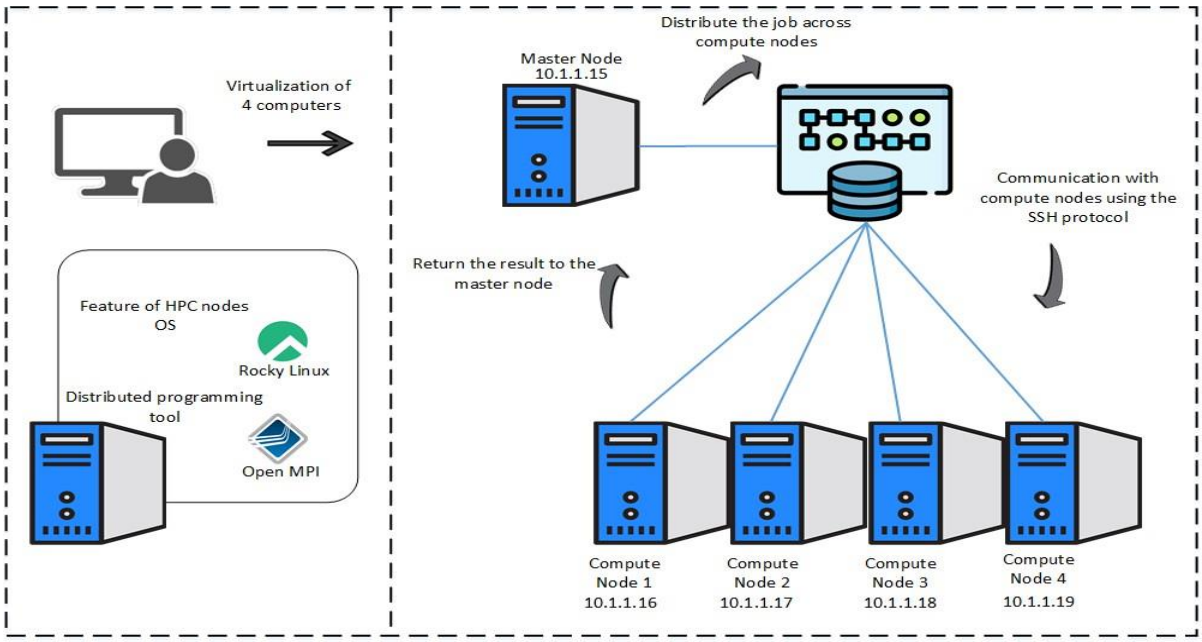
All these needs both vertically sequenced and laterally accommodative computational architecture that could receive, preprocess, and store with equal efficacy. So in such situations, that are a scale above workstations bordering high-performance computing clusters would be the single-stop game changer solution. A high-performance computing facility (FISH@CMFRI) with multi-core computing and expansive scalability of big data analytics/streaming data processing architecture was developed at the Agriculture Knowledge Management Unit (AKMU) of ICAR-CMFRI Kochi.

# Hardware Specification

| | |
|---|---|
| Computing Capacity | 3 TFLOPS |
| No. of Compute Nodes | 4 |
| Total No. of Processors | 8 |
| Total No. of Cores | 96 |
| Memory per node | 64 GB |
| Total RAM | 256 GB |
| Total Usable Storage Capacity | 4 TB |
| Primary Interconnect | 10 Gbps Ethernet |
| Administration & Management Network | 1 Gbps Ethernet |
| Operating System | Rocky Linux 8.8 |

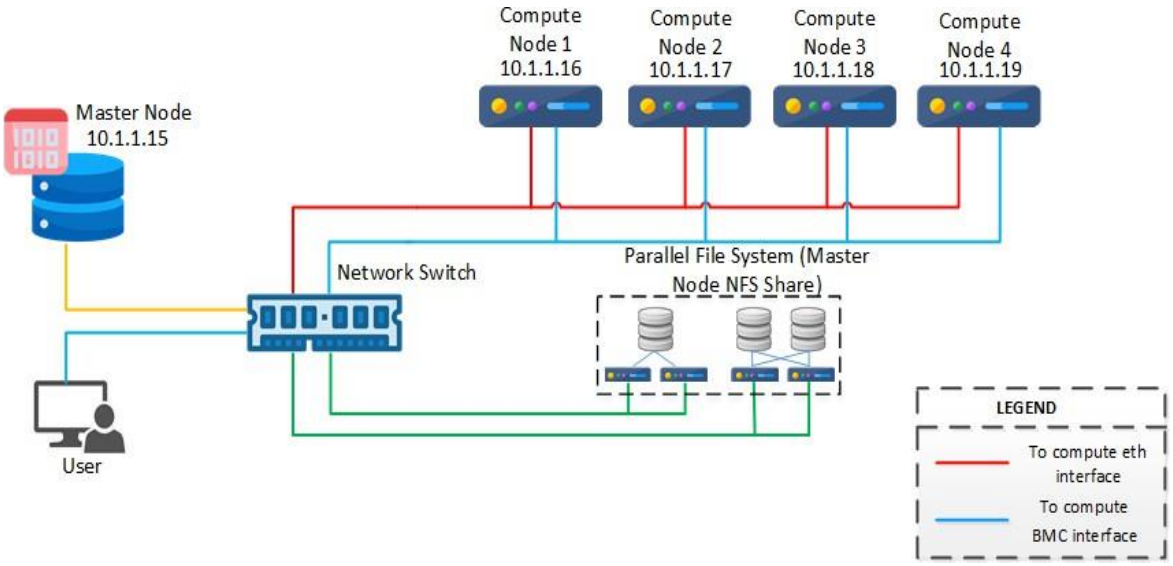
# FISH@CMFRI

FISH@CMFRI HPC solution implemented to perform research and development activities with Rocky Linux 8.8 and OpenHPC stack.




OpenHPC is a collaborative, community effort initiated from a desire to aggregate a number of common ingredients required to deploy and manage High-Performance Computing (HPC) Linux clusters including provisioning tools, resource management, I/O clients, development tools, and a variety of scientific libraries. Packages provided by OpenHPC have been pre-built with HPC integration in mind with a goal to provide reusable building blocks for the HPC community. The master host connected to the local data center network is used to provision and manage the cluster backend. All 4 compute nodes are connected to the master node with local data center switch and these nodes are PXE booted with Rocky Linux 8.8 OS from the master node. Also, there is a shared NFS drive from the master node for running parallel processes across all nodes.

## Configuration of FISH@CMFRI




### 1. User Details

User Name: user01

User Name: user02

User Name: user03

User Name: user04

User Name: user05

(Passwords will be provided on demand).

### 2. Nodes

| | |
|---|---|
| Management Node | 10.1.1.15 |
| Compute-node1 | 10.1.1.16 |
| Compute-node2 | 10.1.1.17 |
| Compute-node3 | 10.1.1.18 |
| Compute-node4 | 10.1.1.19 |

### 3. Compiler Details

Compiler: OpenMPI 4.1.5

# Basic Linux commands

| Command | Description |
|---|---|
| **ls** | The ls command displays list of files in human readable format. |
| **rm** | The rm command Remove files or directories. |
| **cp** | The cp command copies file from source to destination preserving same mode. |
| **cd** | The cd command (change directory) takes us to the destination directory. |
| **cat** | cat command can be used to display the contents of a single or multiple files |
| **pwd** | pwd command return the present working directory |
| **mkdir** | Create a folder or a directory. |
| **rmdir** | Delete a folder or a directory. |
| **touch** | The touch command is used to create a file. It can be anything, from an Empty txt file to an empty zip file. For example, “touch new.txt”. |
| **cp** | Use the cp command to copy files through the command line. |
| **mv** | Use the mv command to move files through the command line. |
| **locate** | The locate command is used to locate a file in a Linux system. |
| **nano, vi** | nano and vi are already installed text editors in the Linux command line. |
| **sudo** | A widely used command in the Linux command line, sudo stands for "Super User Do". So, if you want any command to be done with administrative or root privileges, you can use the sudo command. |
| **df** | Use the df command to see the available disk space in each of the partitions in your system. |
| **ping** | Use ping to check your connection to a server. |
| **Ctrl+c Ctrl+z** | Ctrl+C can be used to stop any command in terminal safely. If it doesn't stop with that, then Ctrl+Z can be used to force stop it. |
| **clear** | You can use the clear command to clear the terminal if it gets filled up with too many commands. |
| **zip, unzip** | Use zip to compress files into a zip archive, and unzip to extract files from a zip archive. |

# Job Scheduler

The HPC Job Scheduler Service is a software program that runs on the head/management node of a high-performance computing (HPC) cluster. It manages the jobs and tasks that are submitted to the cluster, allocating resources to them and monitoring their progress. You can configure the HPC Job Scheduler Service to control how resources are allocated and how jobs are handled. The job scheduler used at CMFRI is PBS pro. PBS stands for Portable Batch System, and it is a software system that is used to manage jobs on HPC clusters.

To run a parallel job using PBS script, you first need to create a PBS script. A PBS script is a text file that contains instructions for the PBS job scheduler. The PBS script will specify the following:

- The name of the job
- The number of nodes that the job needs
- The number of processors per node
- The amount of memory that the job needs
- The command that the job should run

Once you have created the PBS script, you can submit it to the PBS job scheduler using the qsub command. The qsub command will queue the job and the PBS job scheduler will start running it as soon as a set of nodes with the required resources becomes available.

The PBS job scheduler will monitor the progress of the job and will send you email notifications if the job fails or if it exceeds its allotted resources. When the job is finished, the PBS job scheduler will send you a notification that the job has completed.

# How to Access HPC

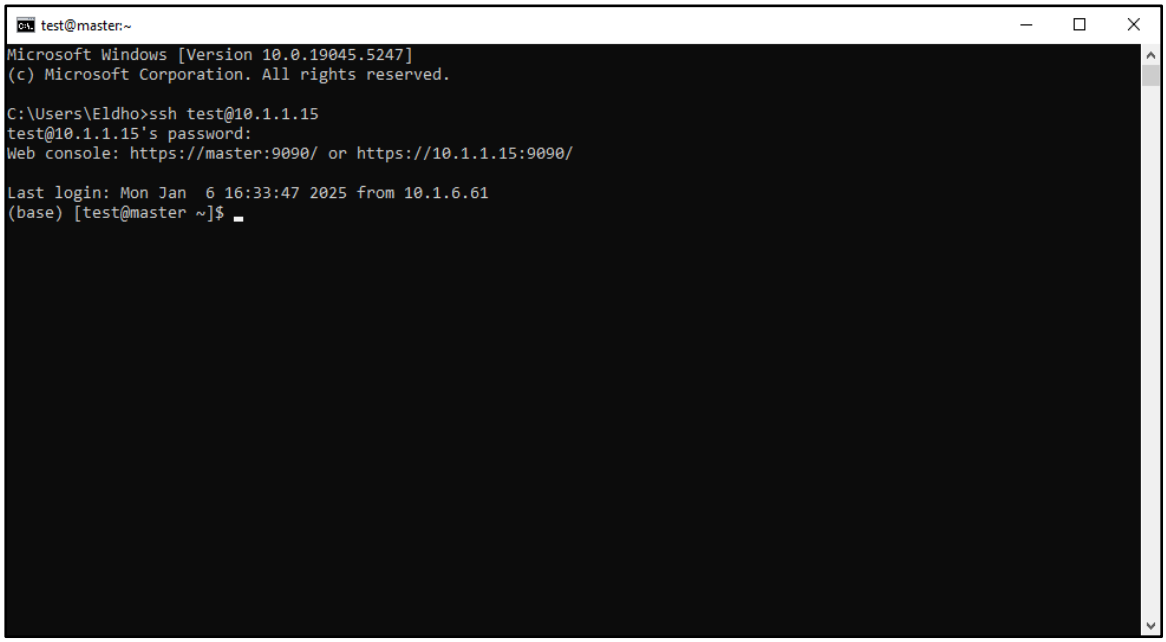
To Access or login to the HPC can be done using SSH (Secure Shell). You can use software like Putty or directly access SSH from Terminal / command prompt.

## 1. Connecting from Terminal/ Command line

Open Command Prompt, type

```
ssh username@10.1.1.15
```

When prompted enter your password and press Enter again. If the connection is successful, you will see a window like this.




** Here **test** is the username, you must enter your username.*

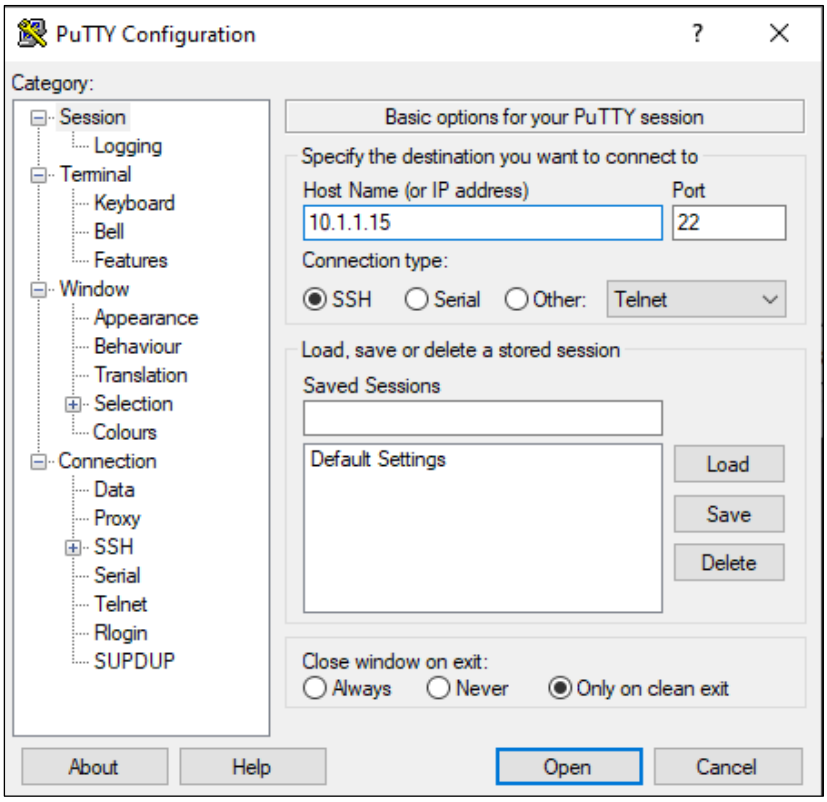
## 2. Connect using Putty

** If you already have access via Command Prompt, you can skip this step.*

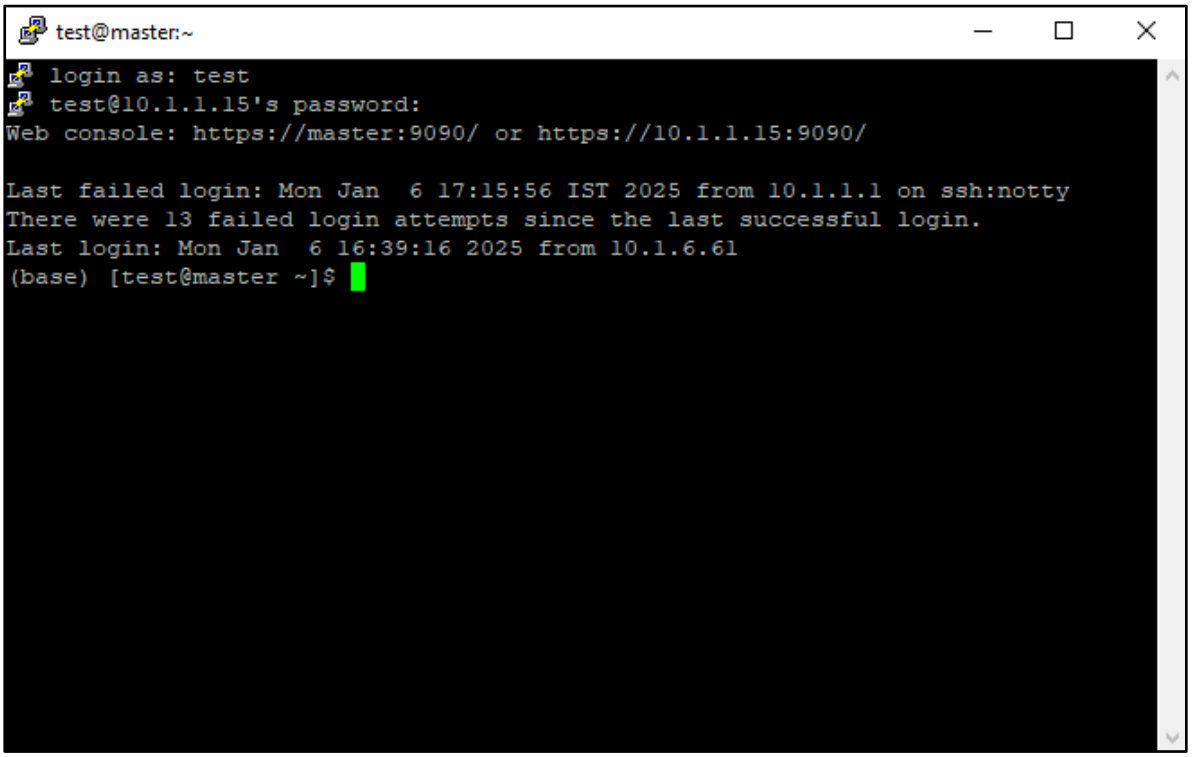
Putty software is available in Microsoft store; the link for the software is given below.

https://apps.microsoft.com/detail/xpfnzksklbp7rj?hl=en-US&gl=IN

After Installation open the App and type the host name as 10.1.1.15 and click open




If any security alert comes you can either choose to *Accept* or *Connect once*. After that enter your username and password.



# How to Run C Code in HPC

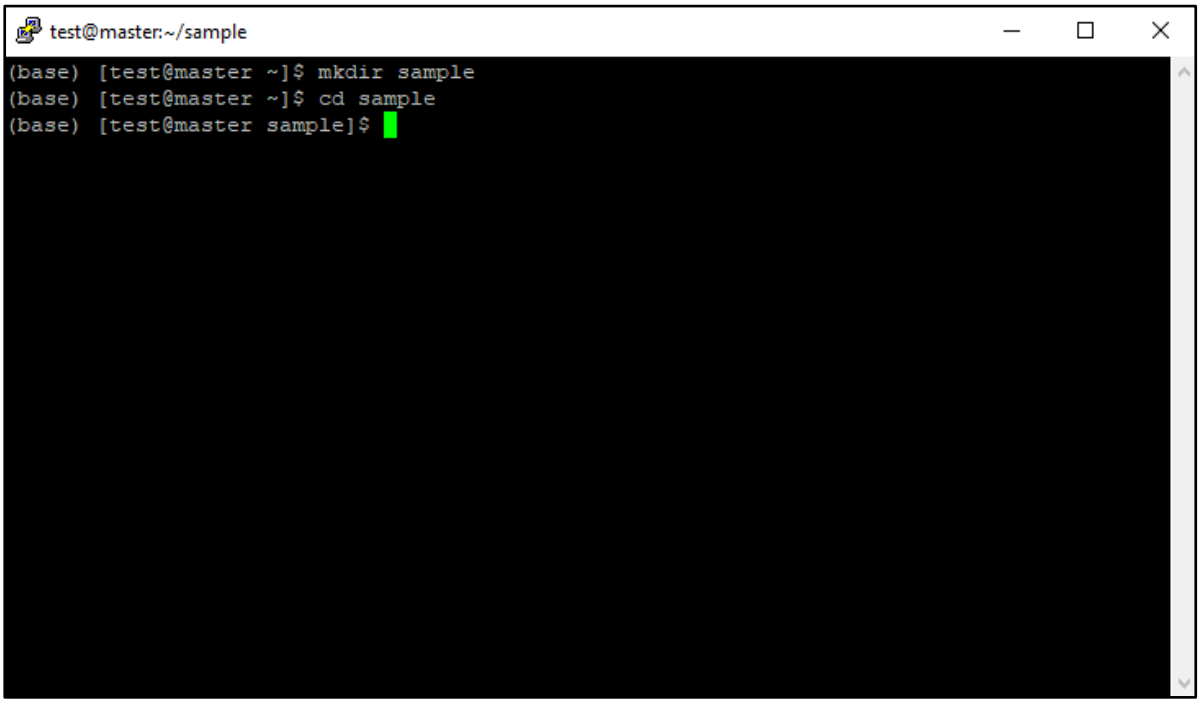
## Creating a C program file

C program files can be created using editors such as vi or nano. Before creating a file first create a folder using the command.

```
mkdir foldername
```

To enter the folder type

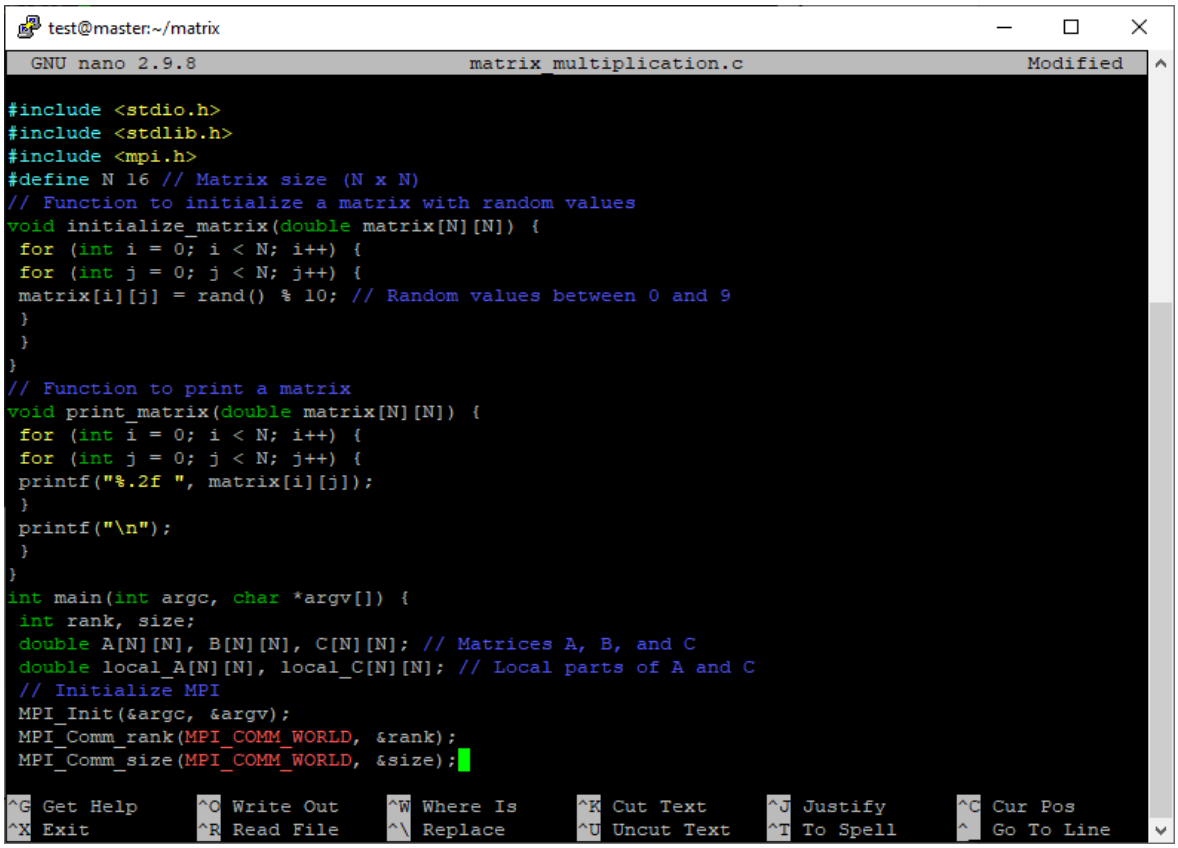
```
cd foldername
```



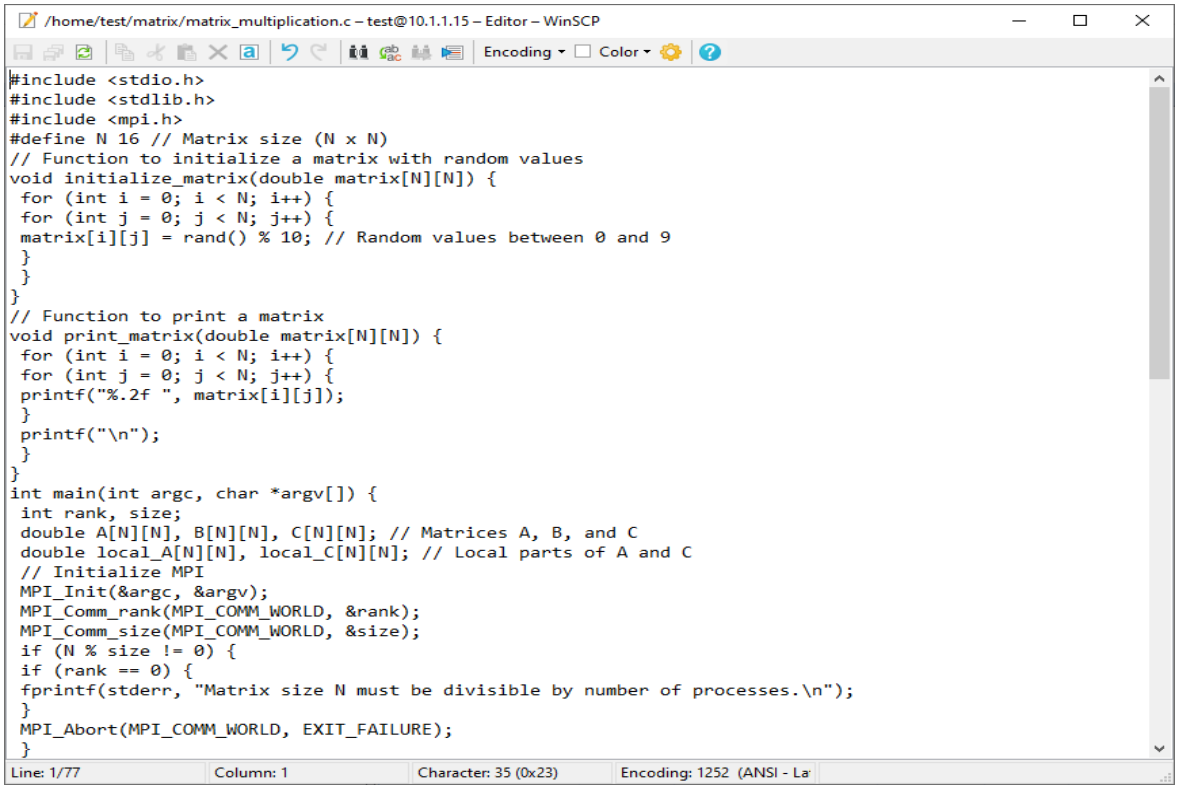

To create a c program file type

```
nano matrix_multiplication.c
```

A text editor will open like this, here you can either type your code or copy code from somewhere else. Just press right click to paste your code.




To save the code press Ctrl+S and to Exit the editor press Ctrl+X.




Full code available in the appendix section.

A C program must be compiled before running. To compile the code, type

```
mpicc -o matrix_multiplication matrix_multiplication.c
```

**mpicc** - This is the MPI compiler wrapper for C programs. It ensures that your program is compiled with the appropriate MPI libraries and headers.

**-o matrix_multiplication -** The -o flag specifies the output filename for the compiled program. In this case, the compiled binary will be named matrix_multiplication

**matrix_multiplication.c**: This is the input source file, i.e., the C code you wrote that contains the MPI program.

While it is possible to submit programs directly to HPC it is generally preferable to create a job script (PBS script). The job script is just like any other code that contains commands that the user would like to run. You can create a PBS script just like how we created c program file, please ensure that the script file has a ".pbs" extension.

### Creating a PBS script file

```
nano matrix_multiplication.pbs
```

Sample PBS script is given below.

```
#!/bin/bash

#PBS -N matrix_mult

#PBS -l nodes=4:ppn=4

#PBS -l walltime=00:05:00

#PBS -j oe

#PBS -o matrix_mult_output.txt


cd $PBS_O_WORKDIR

mpirun -np 16 ./matrix_multiplication
```

**#PBS -N matrix_mult** - This sets the job name to matrix_mult. This name will appear in the job's output files and can be used to identify the job.

**#PBS -l nodes=4:ppn=4** - This request 4 nodes, with 4 processors per node. This means the job will run on 4 compute nodes, each utilizing 4 CPU cores.

**#PBS -l walltime=00:05:00** - This sets a maximum wall time of 5 minutes for the job. If the job exceeds this time, it will be terminated.

**#PBS -j oe** - This directs the standard output and error to the same file, meaning both stdout and stderr will be combined into the same file.

**#PBS -o matrix_mult_output.txt** - This specifies the output file name where the job's combined output (from stdout and stderr) will be stored, which in this case is matrix_mult_output.txt.

**cd $PBS_O_WORKDIR** - This changes the working directory to the directory from which the job was submitted. $PBS_O_WORKDIR is an environment variable that stores the path of the directory the script was run from.

**mpirun -np 16 ./matrix_multiplication** - This command runs the MPI program mpi_matrix_mult using 16 processes in total.

- **mpirun**: This is the command used to launch MPI jobs.
- **-np 16**: This option specifies the number of processes to run, in this case, 16 processes (4 node * 4 processors per node), if we use only 2 nodes and 5 processors per node then it will be 10.
- **./matrix_multiplication**: This is likely an executable that performs matrix multiplication using MPI for parallelization.

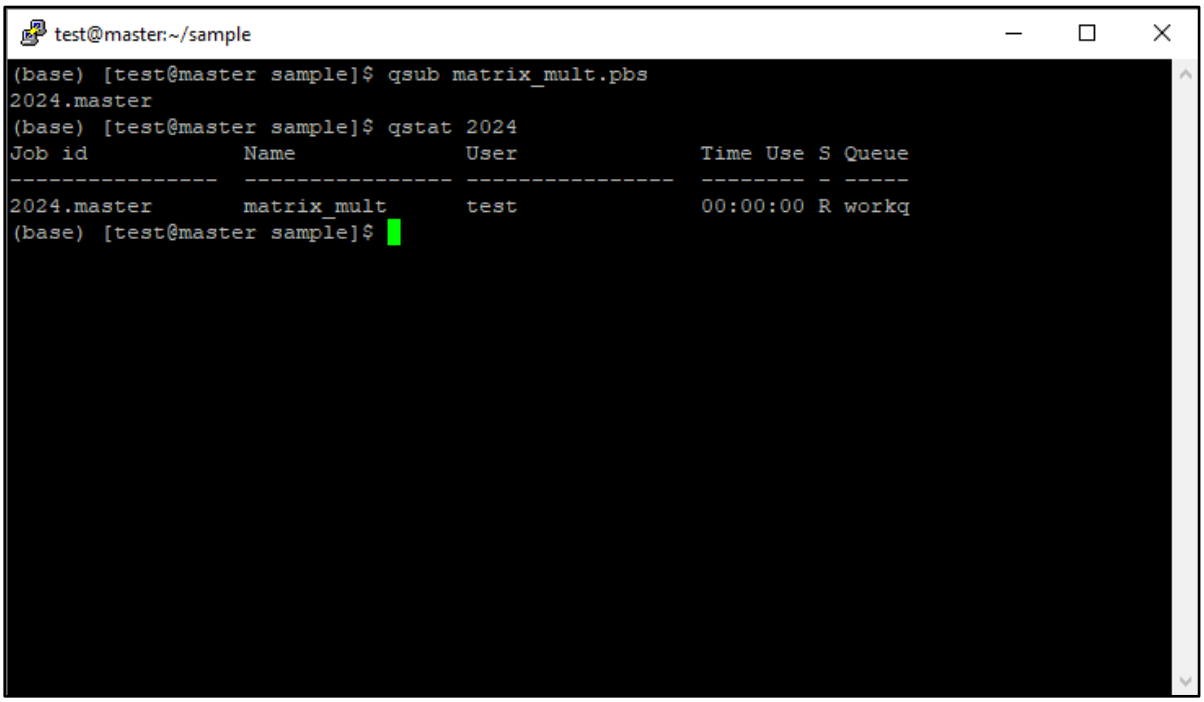
Users can submit their jobs by using qsub command followed by pbs script name.

```
qsub matrix_mult.pbs
```

User can also check the status of the submitted job using qstat command,

```
qstat <Job_id> or qstat -an
```

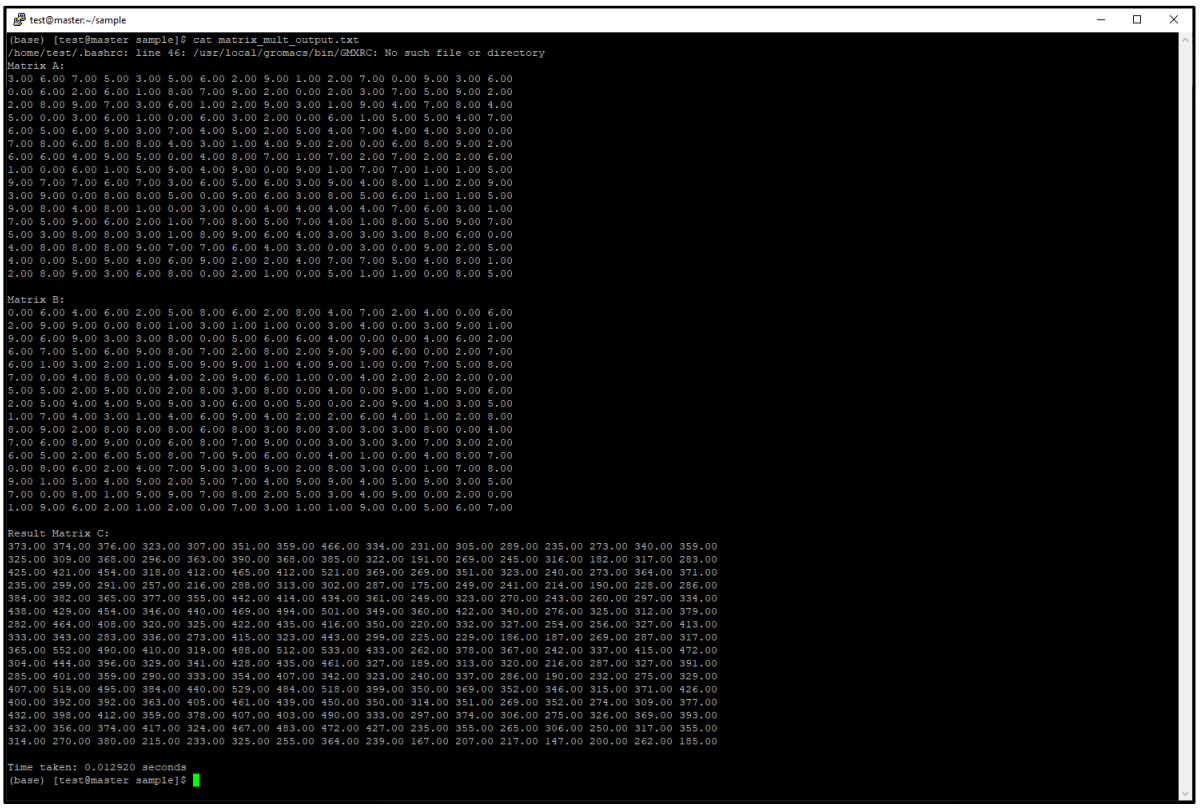
**-an** : Where **a** refers to status of all jobs in the pbs queue and **n** refers to which nodes the jobs are running




**2024 is the Job id PBS assigned to the above job as you can see in the picture.*

After the job completes, users can refer to the output files for result details or error diagnostics. For that type

```
cat matrix_mult_output.txt
```




**Time Taken: 0.012920 seconds*

**Please note that execution time may vary depending on system load, network conditions, and available resources.*

The result above corresponds to a 16 x 16 matrix multiplication. Using the same setup, we achieved a time of 180.214546 seconds for a 10000 x 10000 matrix multiplication. The code used for this calculation is provided in the appendix section, where you can adjust the matrix size (N) up to 10000 to observe the results.

### To cancel a Job

```
qdel <Job_id>
```

# How to Run R Code in HPC

After successfully logging in, you can create an R script just like how we created a C program file. Ensure the script file has an ".R" extension. Unlike a C program, R code does not require compilation. Simply create the R script and a PBS script to run the R code on the HPC.

*sample.pbs* script is given below.

```
#!/bin/bash
#PBS -N R_MPI_parallel_test
#PBS -l nodes=4:ppn=24,mem=64gb
#PBS -l walltime=02:00:00
#PBS -j oe
#PBS -o output.log

module load R
module load gnu12/12.3.0
module load openmpi4/4.1.6

export R_LIBS="/home/test/R/x86_64-redhat-linux-gnu-library/4.4:$R_LIBS"

cd $PBS_O_WORKDIR

mpirun --map-by node -np 96 Rscript sample.R
```

**sample.R script is provided in the appendix section.*

**module load <module-name>** - This command loads the necessary module into your environment on the HPC. For parallel execution in R above three modules are required. It loads R environment, GNU Compiler Collection, which might be required for dependencies, OpenMPI for parallel execution.

**export R_LIBS="/home/test/R/x86_64-redhat-linux-gnu-library/4.4:$R_LIBS" -** Adds a user-specific R package directory to the R library search path.

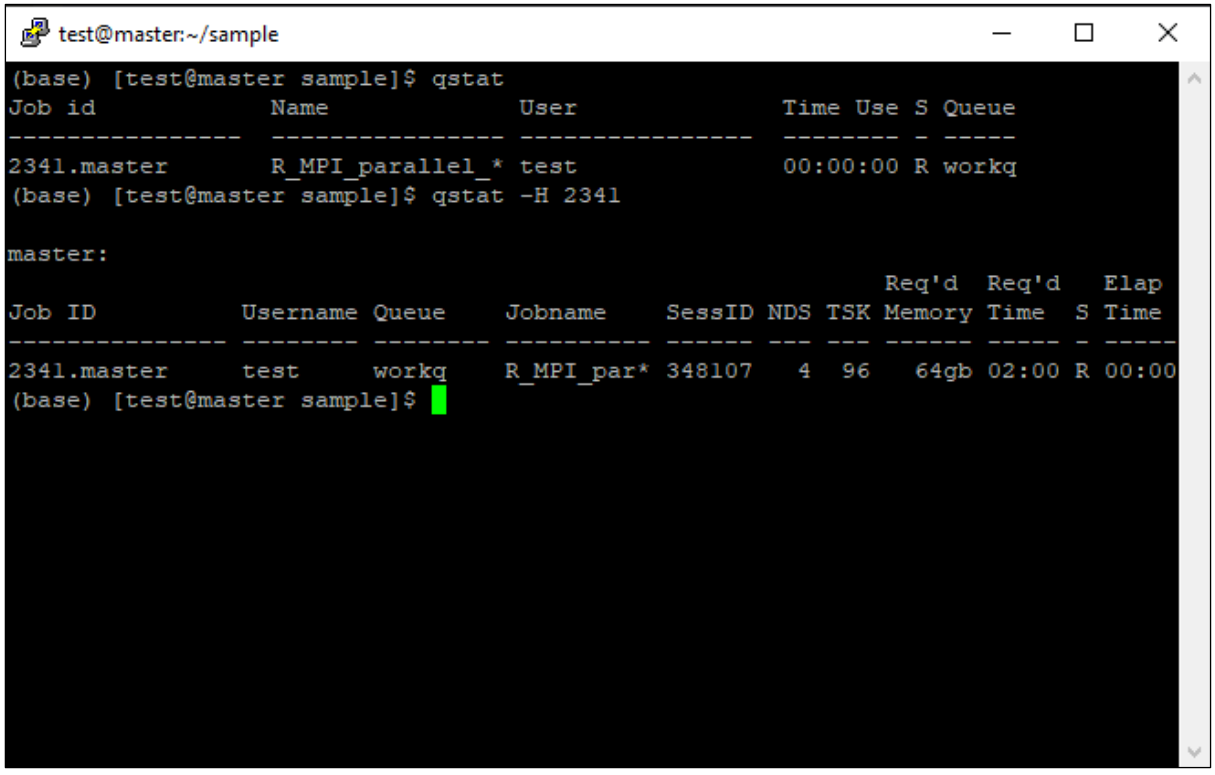
**mpirun --map-by node -np 96 Rscript sample.R** - Runs the R script named *sample.R,* "*--map-by node*" command will force MPI to distribute ranks evenly across nodes.

Users can submit their jobs by using qsub command followed by pbs script name.

```
qsub sample.pbs
```

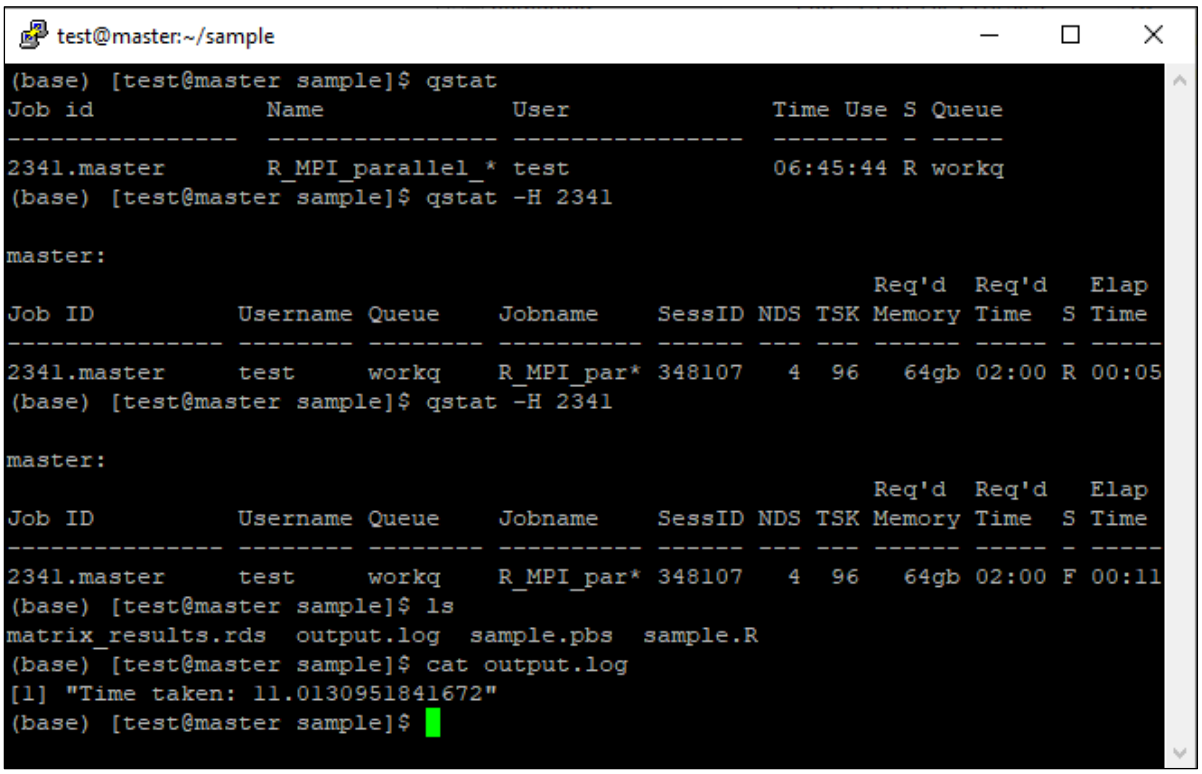
To check the status of the specific job in details type,

```
qstat -H <Job_id>
```




The 'S' column represents the status code of the job, with 'R' indicating that the job is Running. A complete list of status codes can be found in the appendix. After the job finishes, multiple files will be generated based on your code, which can be viewed using the **ls** command. To open a specific file, you can use the **cat** command.

```
cat output.log
```



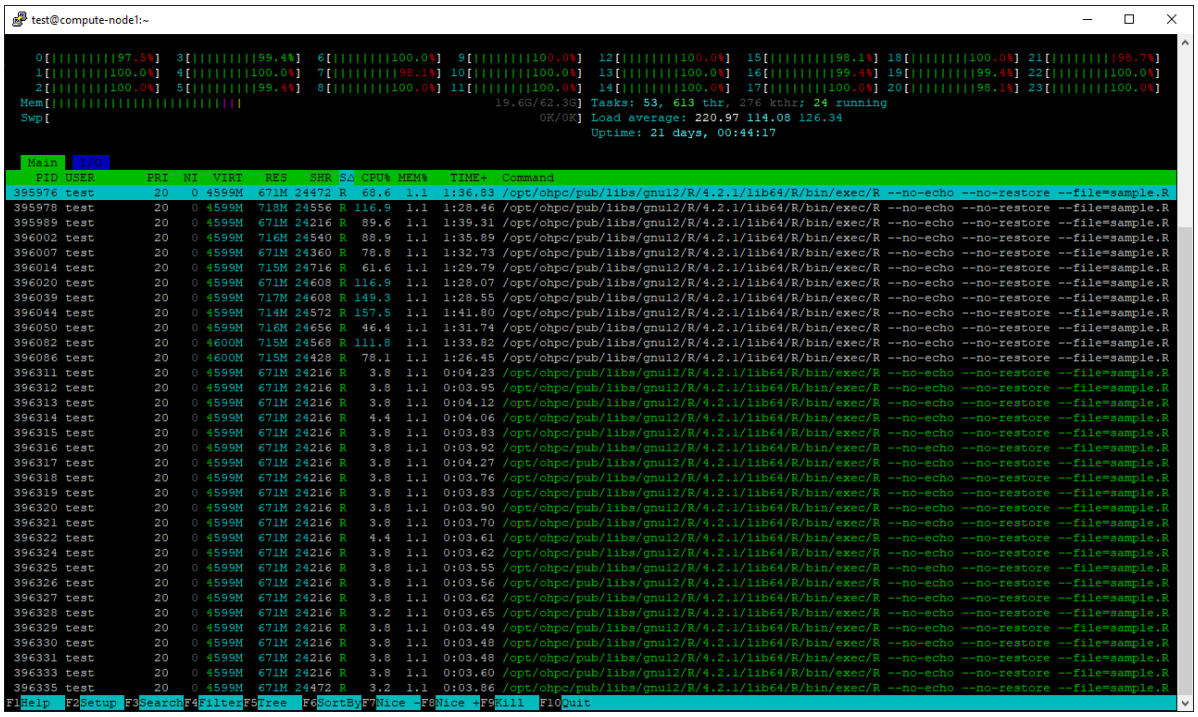

**Time Taken: 11.0130951841672*

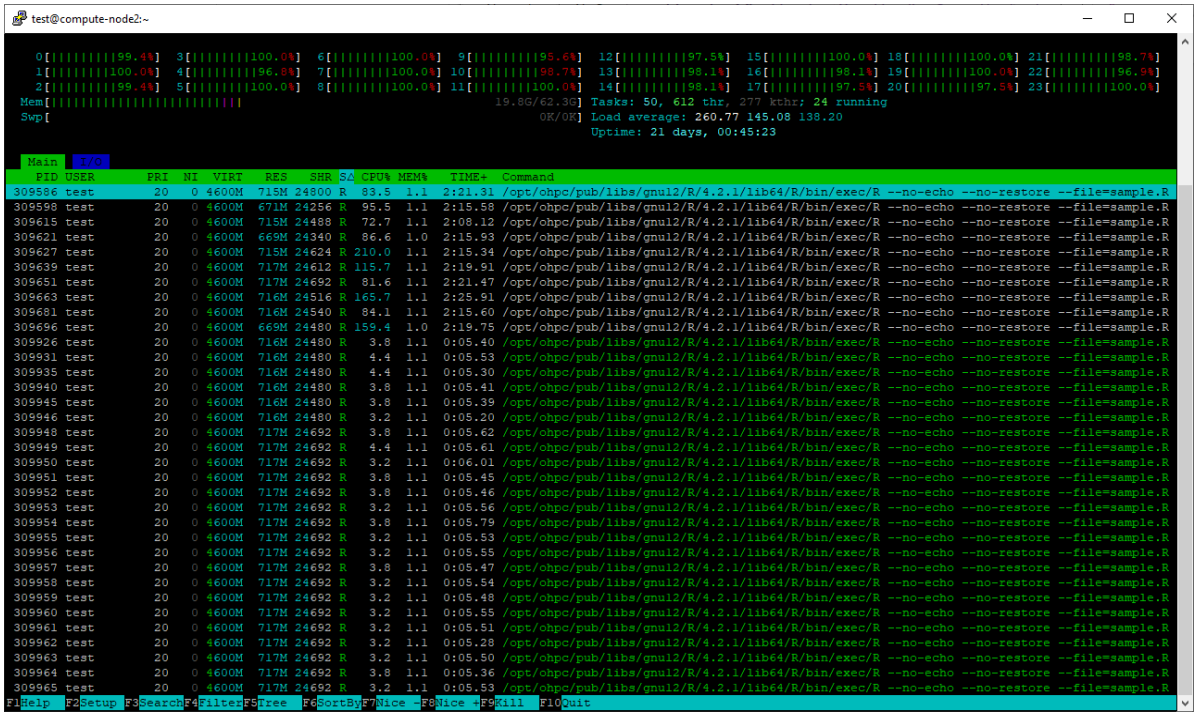
Above R code performs intensive matrix operations and statistical analysis that go beyond the capabilities of a standard computer. If you check the code which is in appendix section, the computed results are saved as an RDS file (matrix_results.rds) for further analysis, which will be discussed in later sections.

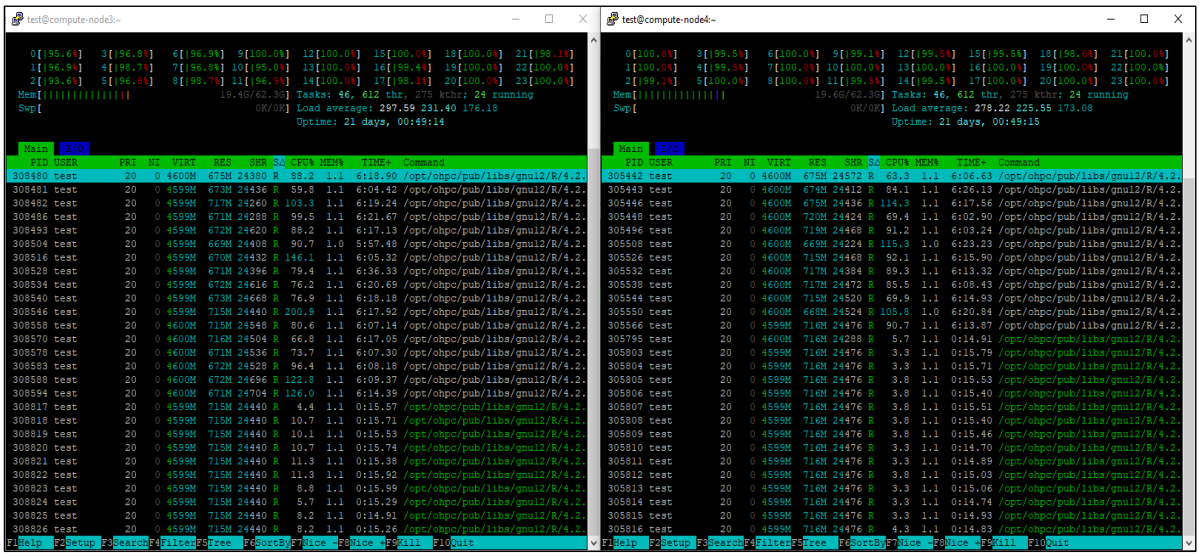
In **R** Parallel Packages like *doParallel, parallel, pbdMPI, doMPI* and *doFuture* uses a common set of instructions to use parallel capabilities as follows:

```
library("package-name")

cl <- makeCluster(NumberofCores)

register_cluster(cl)

... #code to be run in parallel mode

stopCluster(cl)
```

To verify whether the code is running across all four nodes, the **htop** command can be used. It's a popular interactive process monitoring tool in Linux. It provides a real-time overview of system resource usage, including CPU, memory, and running tasks, The below screenshot shows our four compute nodes.






** Logging in to each node is required to view resource usage using the htop command*

## Explanation of the key elements in the image

**CPU Usage:** The numbers 0-23 indicate the logical CPU cores (24 cores total). Each bar represents the usage of an individual core, with 100.0% indicating full utilization.

**Memory Usage:** The *Mem* bar shows the amount of system memory used. The numbers 24.0G/62.3G indicate that 24 GB out of 62.3 GB total memory is currently in use.

**Swap Usage:** The *Swp* section indicates swap memory usage.0K/0K suggests that no swap space is allocated or in use.

**Task Information:** Tasks: 50 indicates there are 50 total processes running.612 thr refers to the total number of threads across all processes.24 running means 24 processes are actively using the CPU.

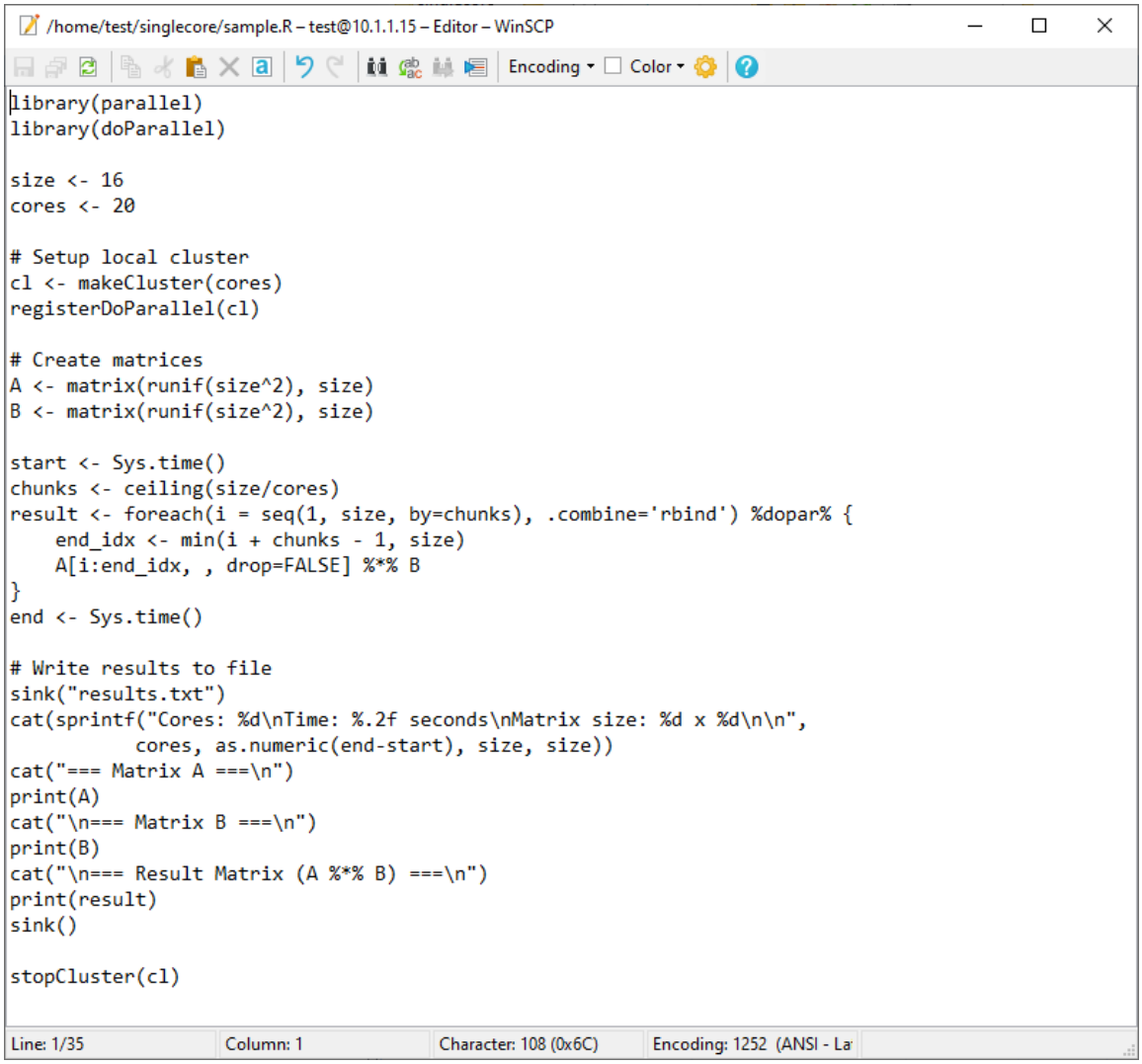
**Load Average:** The values 175.82 263.15 180.74 represent system load averages over the past 1, 5, and 15 minutes, respectively. These values are significantly high, indicating the system is under heavy CPU load.

**System Uptime:** Uptime: 21 days, 00:34:46 shows that the system has been running continuously for 21 days and 34 minutes.

### Running R Code in Parallel on a Single Node

In R, packages like *parallel* and *doParallel* provide efficient parallel computing capabilities, but they are limited to a single node. These packages utilize multi-core processing by dividing tasks across the available cores of a single machine, enabling significant performance improvements for data-intensive computations. However, they do not support distributed computing across multiple nodes in an HPC cluster.

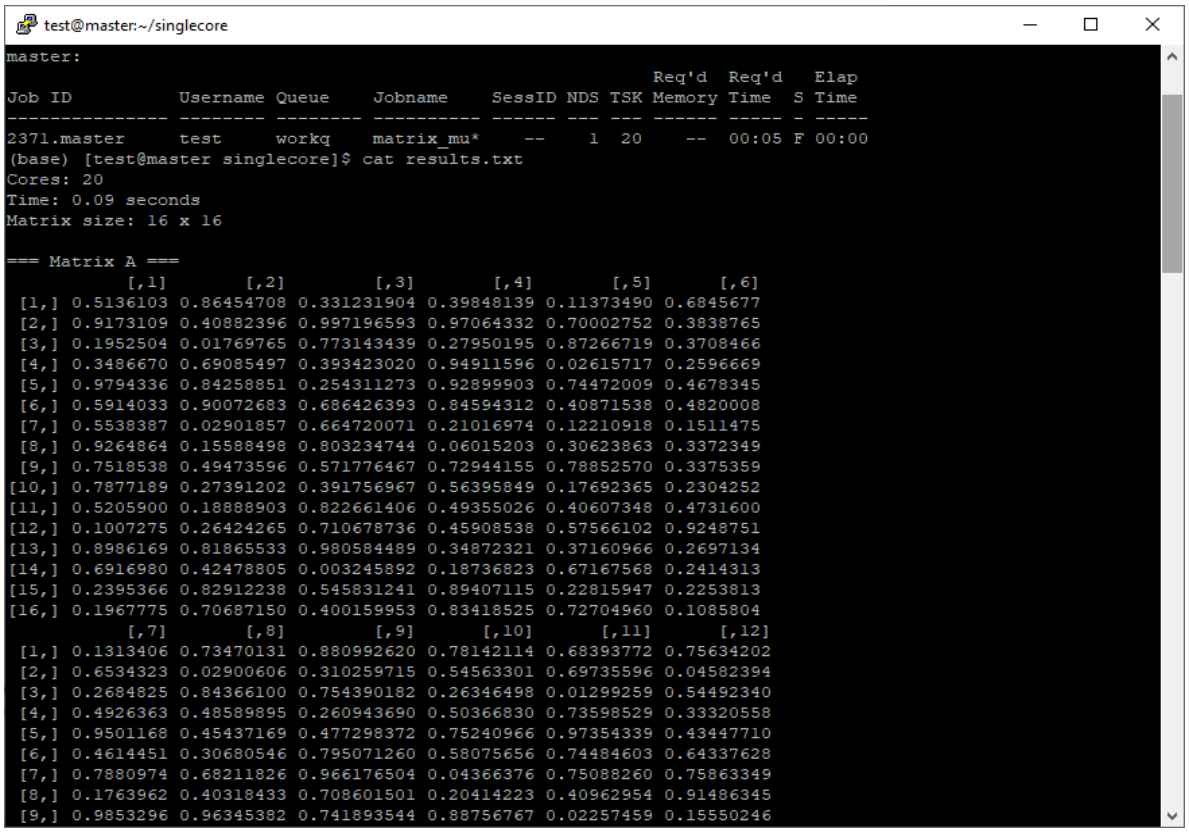
For workloads that require multi-node parallelism, R users need to leverage specialized frameworks such as **MPI (Message Passing Interface)** through packages like *pbdMPI* and *Rmpi* like we used before. R program that uses only single node is given below.



Pbs script to use for the above code is

```
#!/bin/bash

#PBS -N matrix_mult

#PBS -l nodes=1:ppn=20

#PBS -l walltime=00:05:00

#PBS -o matrix_mult_output.txt


Module load R

cd $PBS_O_WORKDIR


R CMD BATCH sample.R
```

Running the code and other processes are same, the output of the program is shown below.



# Transfer Files between Windows and HPC

You can transfer files from your HPC to your Windows machine by using the PuTTY's built-in SCP command (PSCP).

If PuTTY is installed, you can typically find *pscp.exe* in the same directory as PuTTY, such as *C:\Program Files\PuTTY* or *C:\Program Files(x86)\PuTTY*

Open command prompt and use the following command.

```
"C:\path\to\pscp.exe" username@hpc_address:/path/to/your/file.txt
C:\destination\path
```

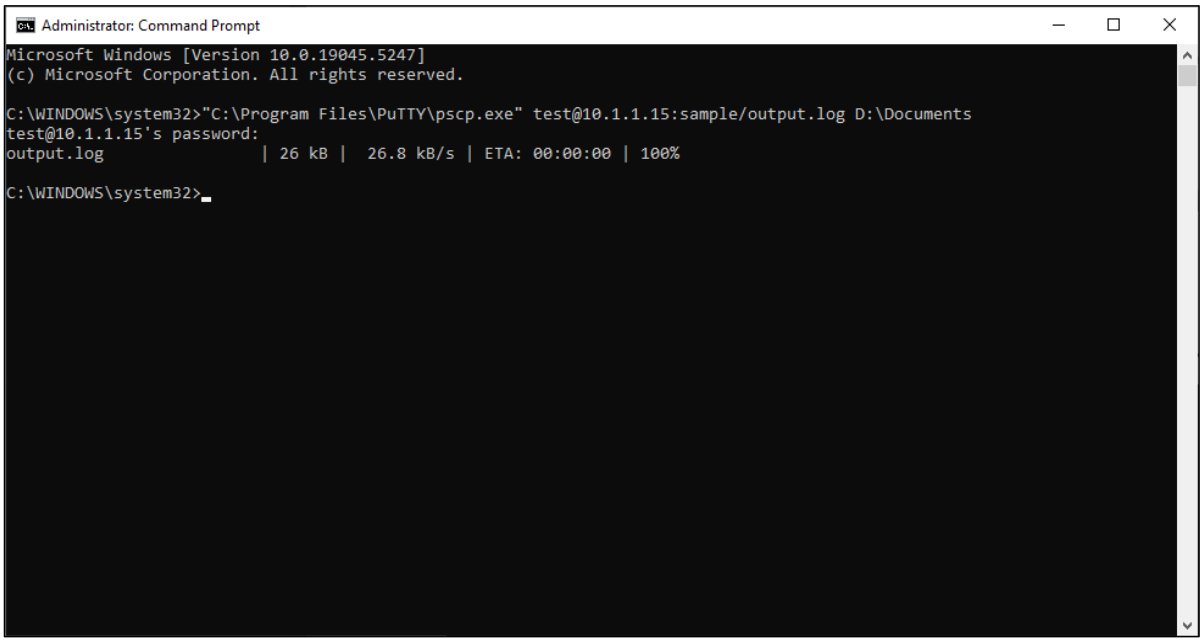
Replace:

- "C:\path\to\pscp.exe" with the full path to pscp.exe on your system.
- Username with your HPC username.
- hpc_address with the server address (e.g. 10.1.1.15).
- /path/to/your/file.txt with the location of the file on the HPC.
- C:\destination\path with the folder path on your Windows machine.

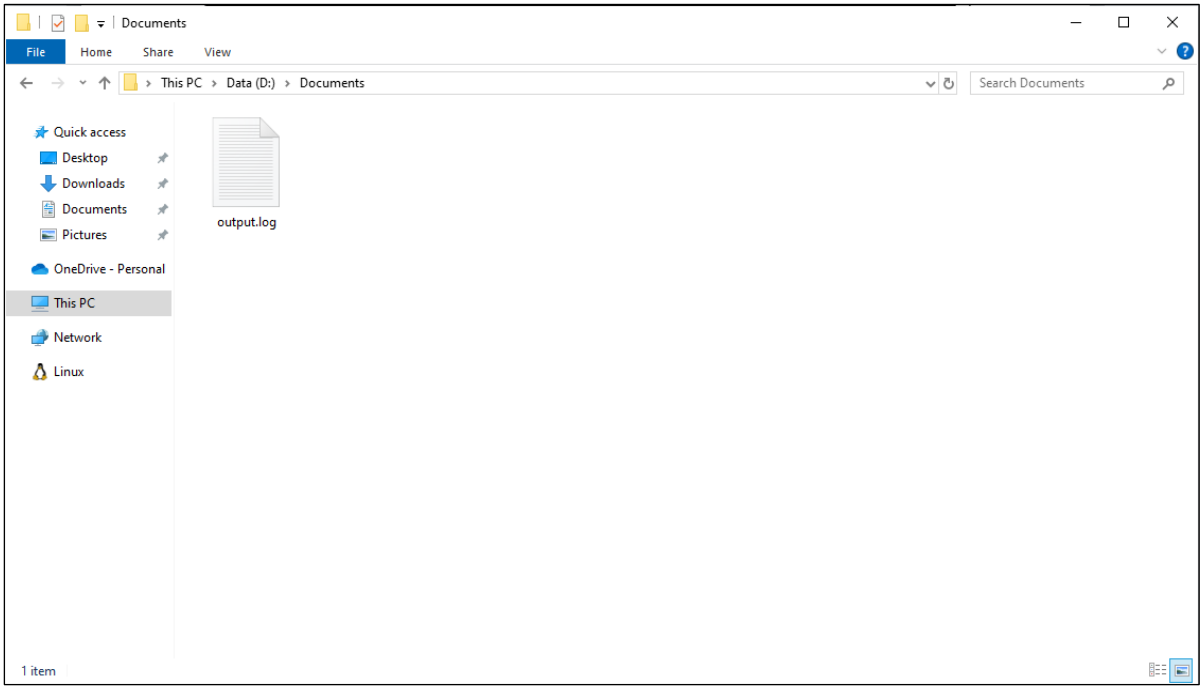
Example

```
"C:\Program Files\PuTTY\pscp.exe" test@10.1.1.15:sample/output.log
D:\Documents
```

After executing the command, it will be prompted to enter your password for authentication. Once authenticated, the file will be downloaded to the specified directory. Ensure the file name is correctly typed, including capitalization (Linux paths are case-sensitive).



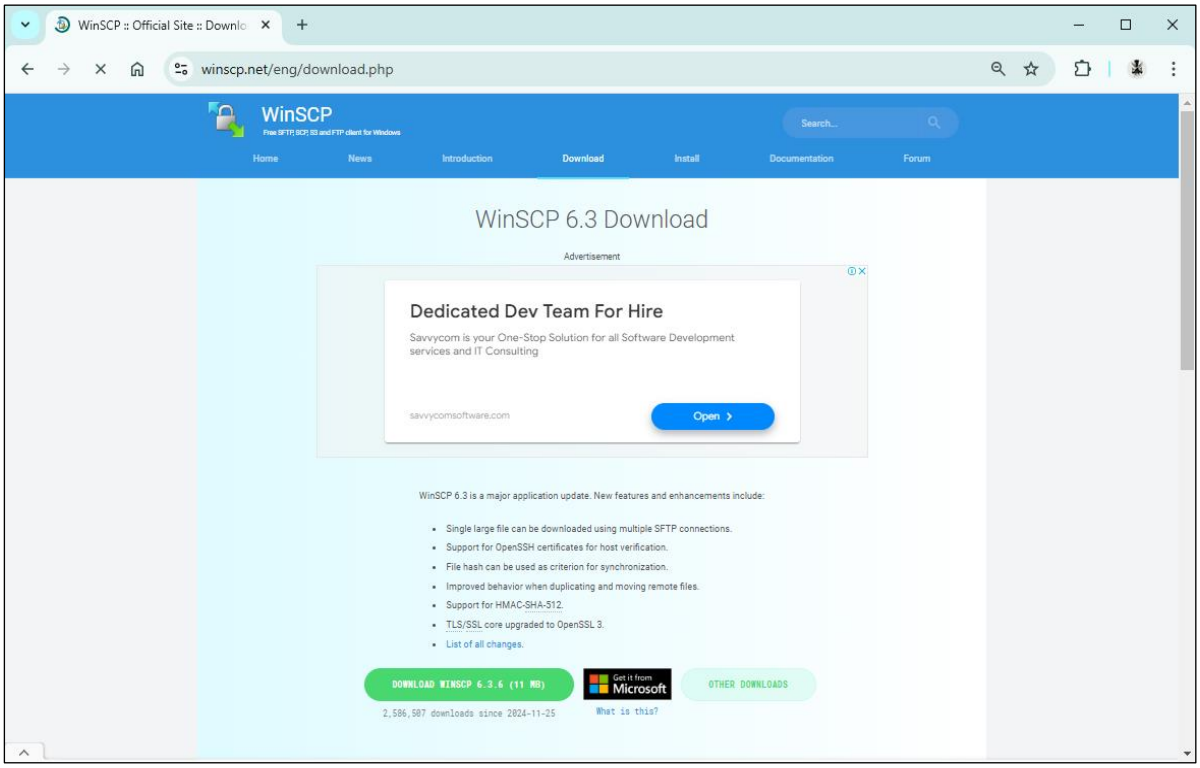

**Wait for the file to download 100% before opening in your system.*



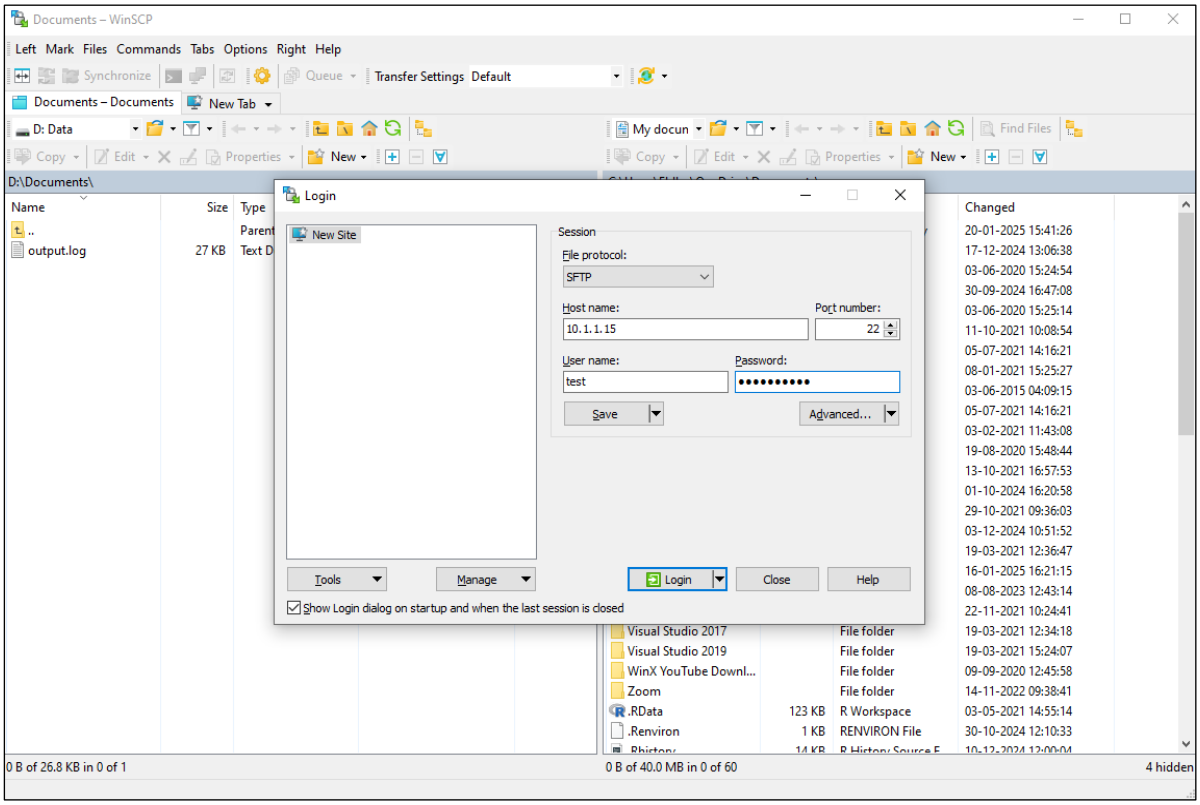

After downloading, you can see the files in your destination folder.

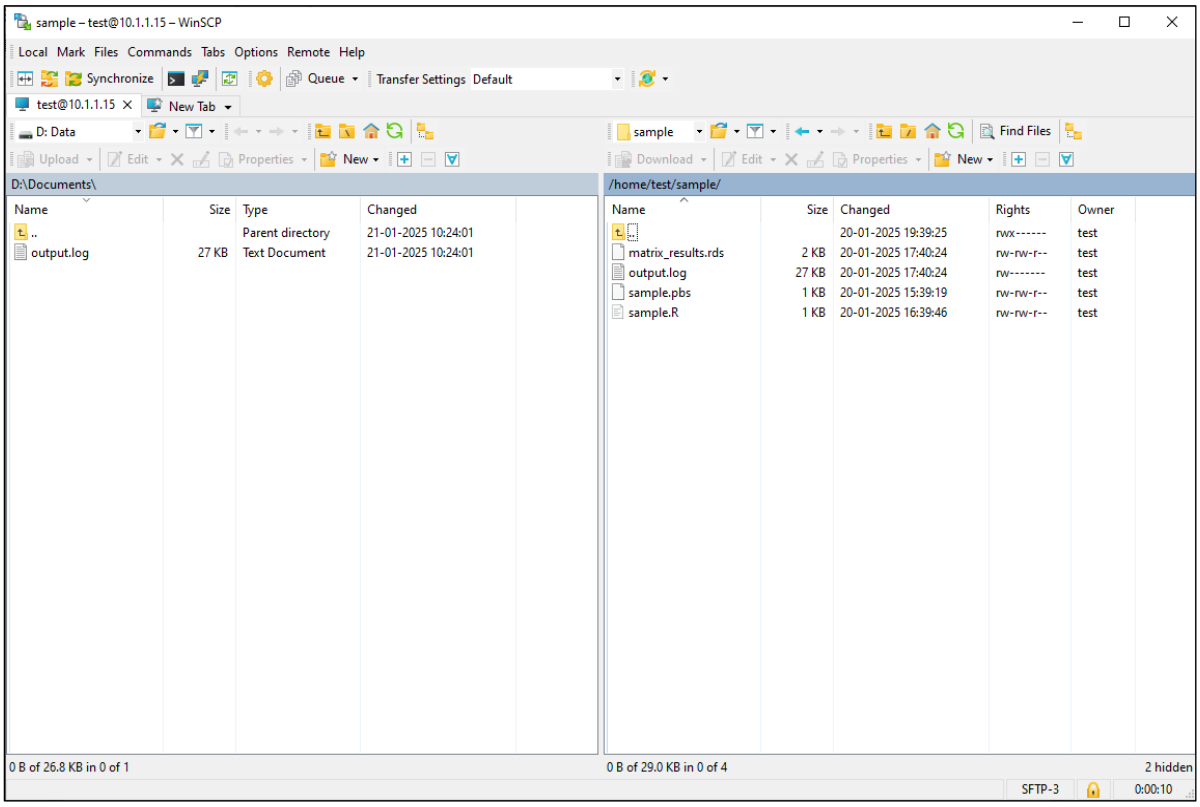
If you're not comfortable using the command line interface, you can use GUI-based software to transfer files between the two systems. Below are the instructions for using those tools.

Various tools such as WinSCP and FileZilla, are available for transferring files between HPC and Windows systems. Below are some screenshots of the WinSCP software.

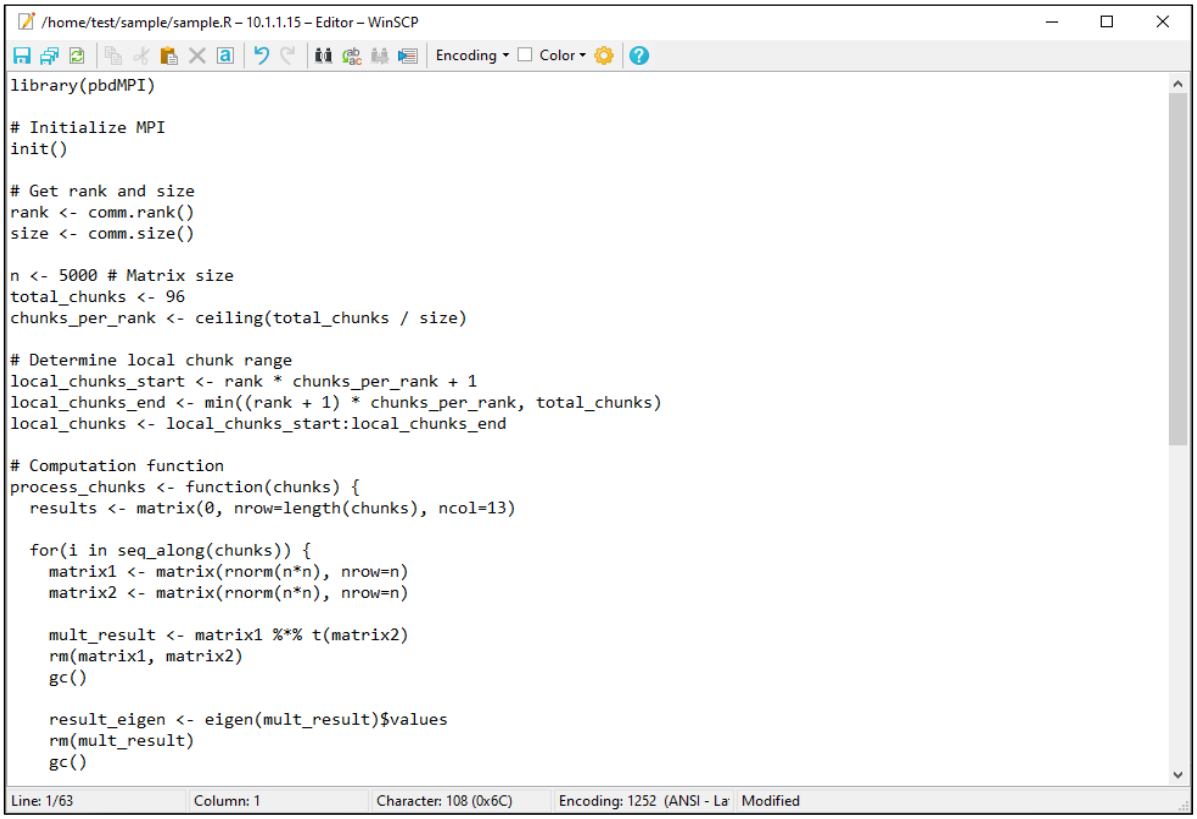



**Software download link is given in the Additional Resources section.*






The left panel displays your Windows system files, while the right panel shows the HPC files. You can drag files between the panels or double-click to view or edit them.



### Open R environment inside HPC

There is an alternate way to access R environment inside the HPC system. After successfully logging in:

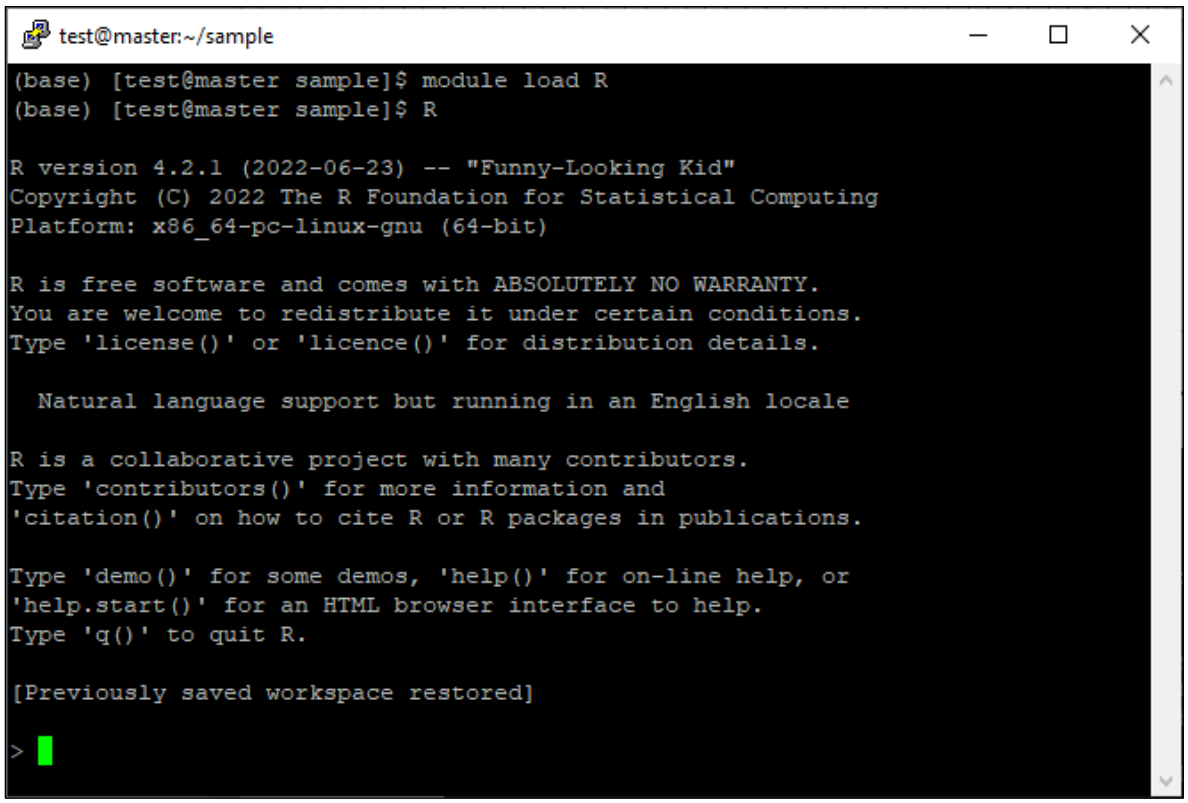
Start the R session by typing the following commands:

```
module load R
```

The first command loads the R module.

```
R
```

The second command starts the interactive R session. It will display the R version and related information shown in the picture below.




### To install a package

Before installing a package, you must set path to the shared storage, so that the package can be used by all the nodes in the hpc. To set path use this command.

```
.libPaths("/home/test/R/x86_64-redhat-linux-gnu-library/4.4")
```

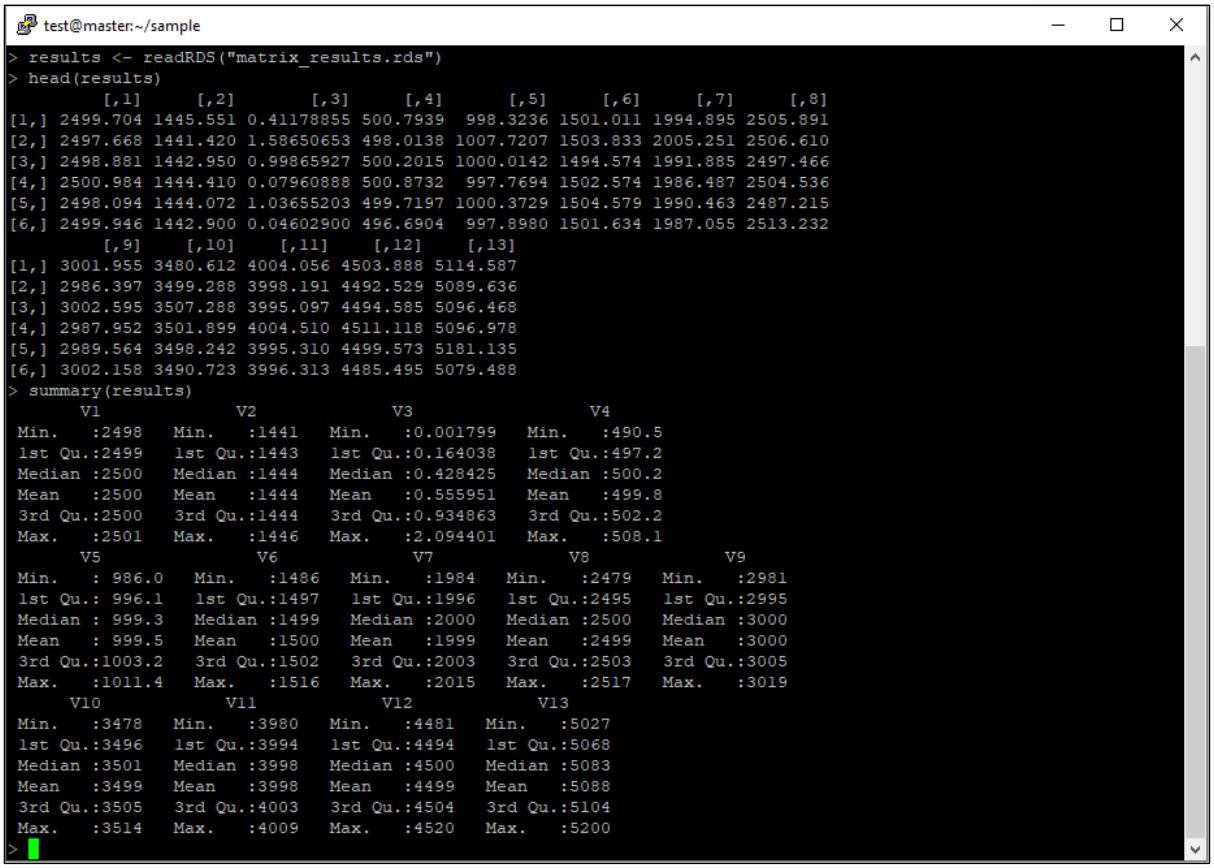
After that you can install any package using the below command.

```
install.packages("package_name")
```

During the installation process, R will prompt you to select a CRAN mirror for downloading packages. You can choose either "42: India (Bengaluru)" or "43: India (Bhubaneswar)". To proceed, simply type **42** and press **Enter**.

You can view the results saved from the *sample.R* script by loading the .rds file using R, follow these steps to see the results.

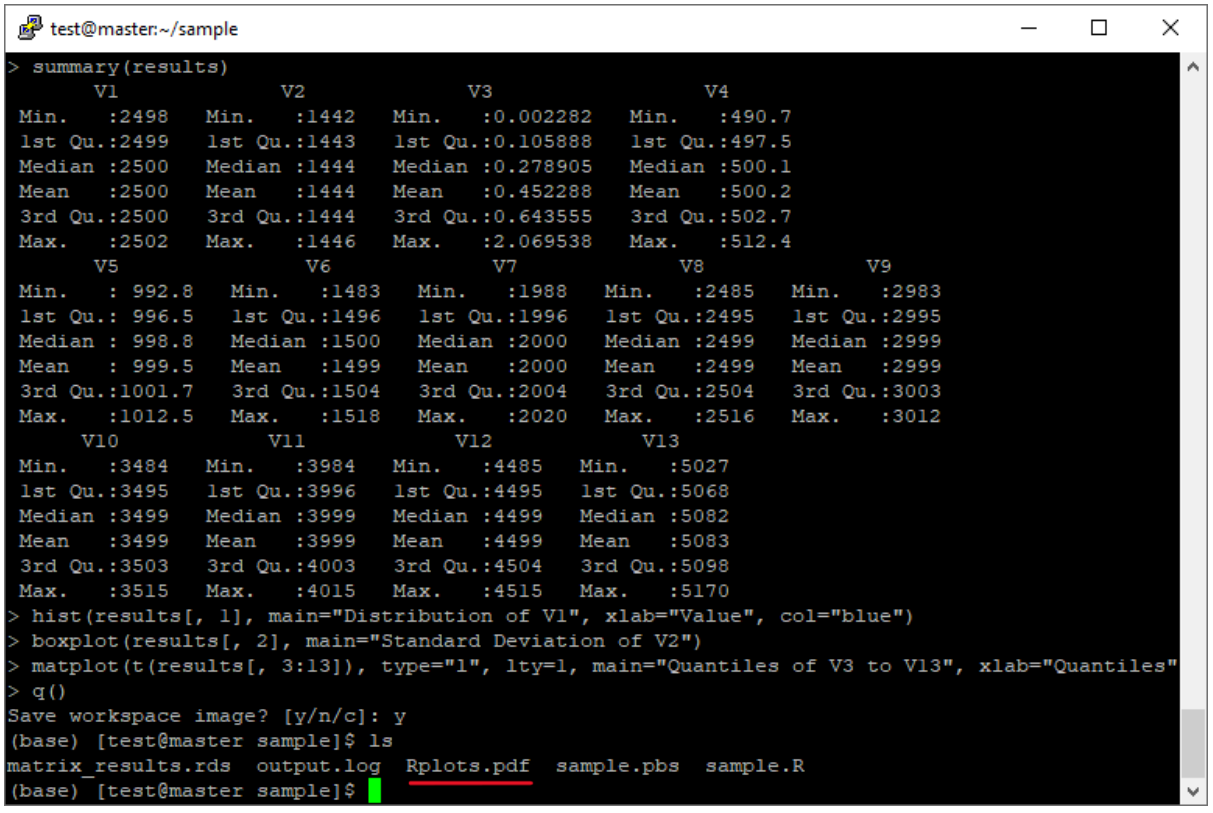
```
results <- readRDS("matrix_results.rds")

# View the first few rows of results
head(results)

# Summary statistics
summary(results)
```



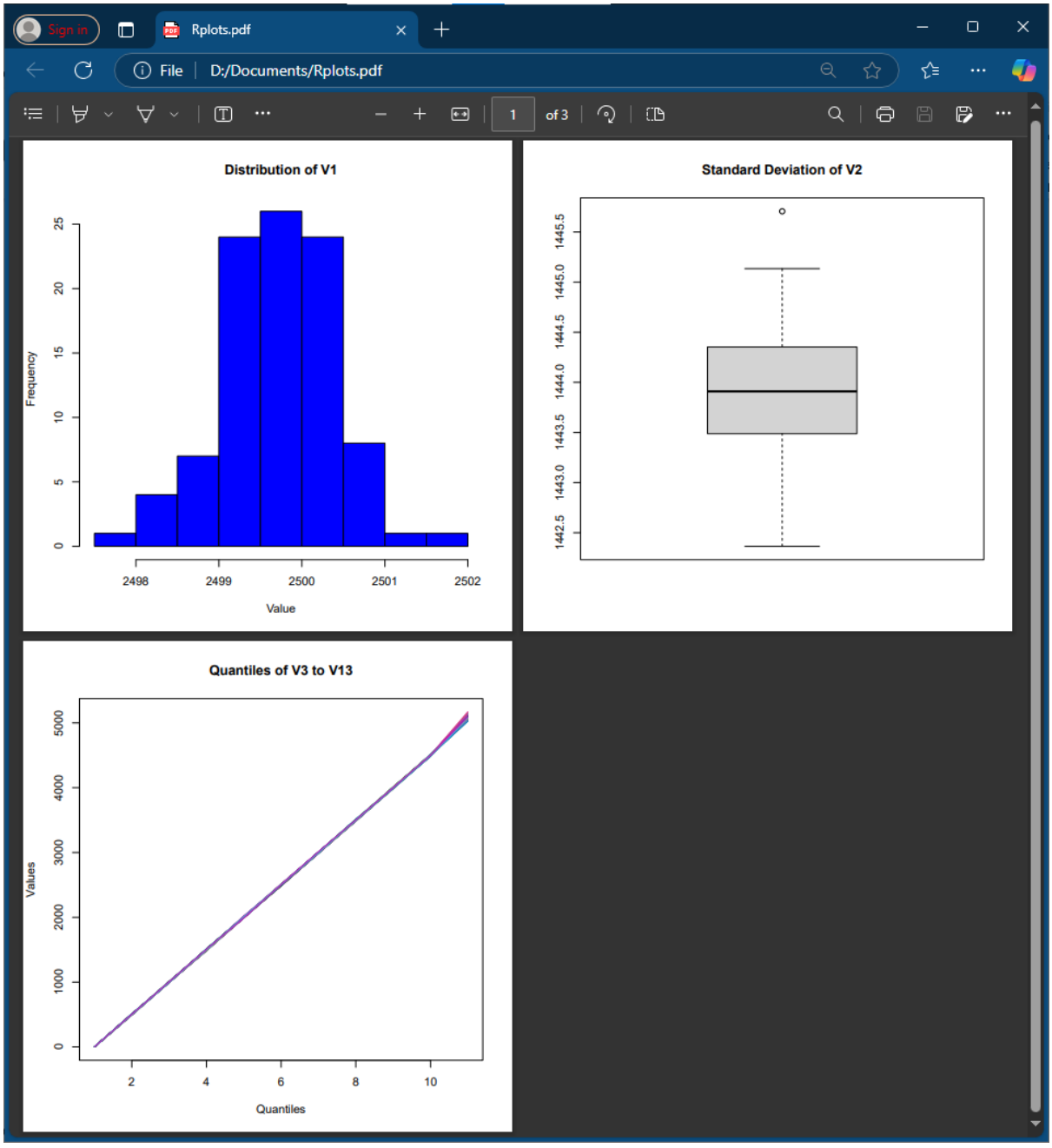
You can visualize the computed values using plots

```
# Histogram of column V1
hist(results[, 1], main="Distribution of V1", xlab="Value", col="blue")

# Boxplot of standard deviation
boxplot(results[, 2], main="Standard Deviation of V2")

# Plot quantiles (columns 3 to 13)
matplot(t(results[, 3:13]), type="l", lty=1, main="Quantiles of V3 to V13", xlab="Quantiles", ylab="Values")
```

After executing these commands, the console will not produce any output. Once you exit the R environment, a new file called **Rplots.pdf** will be generated in the current directory.



You can download the file to your windows system using any method and check the result.




## Exiting the R Environment

To exit the R session when finished, simply type

```
q()
```

# Good Practices for Using HPC

**Request Resources Wisely**: Only request the CPU, memory, and wall time your job needs to ensure efficient use of the system.

**Avoid Login Node Overload**: Do not run resource-intensive tasks on login nodes; always submit jobs through the scheduler.

**Clean Up Regularly**: Remove unused files and manage your storage to stay within quota limits.

**Optimize Code Performance:** Profile and optimize your code to improve efficiency, minimize bottlenecks, and reduce runtime.

**Monitor Jobs**: Check your job logs and resource usage to optimize performance and identify issues.

**Test Small, Scale Gradually**: Run small test jobs before submitting large-scale jobs to estimate resource needs and debug issues.

**Use Parallelization Effectively:** Utilize parallel programming techniques (e.g., MPI, OpenMP, multi-threading) to make the best use of available cores and nodes.

**Report Issues Promptly:** Notify the HPC team if you encounter persistent issues, unexpected failures, or performance bottlenecks.

**Follow Fair Use Policies**: Use resources responsibly and share the HPC system fairly with other users.

# FISH@CMFRI Power-Off Procedure

### Stop PBS Jobs Gracefully

Log in to the head node and prevent new jobs from starting. You can do this by setting the nodes to an offline state.

```
qmgr -c "set server scheduling = false"
```

Check for running jobs and wait for them to finish, or you can choose to cancel the jobs. After that set all nodes offline.

```
qmgr -c "set node compute-node[1-4] state=offline"
```

### Shutdown Compute Nodes

Once jobs have stopped and nodes are offline, initiate the shutdown of compute nodes.

```
pdsh -w compute-node[1-4] "sudo shutdown now"
```

For all compute nodes, use

```
pdsh -a "sudo shutdown now"
```

### Stop PBS Services

Stop the PBS server on the head node.

```
sudo systemctl stop pbs
```

### Shutdown Head Node

After all services are stopped, shut down the head node.

```
sudo shutdown now
```

# FISH@CMFRI Power-On Procedure

### Power On Networking and Storage

Start by powering on the networking equipment (switches, routers), followed by storage systems like NAS or other shared storage devices.

### Power On Head Node

Power on the head node and wait for it to boot fully.

### Start PBS Services

Once the head node is online, start the PBS services.

```
sudo systemctl start pbs.service
```

### Power On Compute Nodes

Power on the compute nodes using IPMI, a remote management tool, or manually.

### Mount Shared File systems

On the Head node, ensure the shared file systems are mounted.

```
sudo mount -a
```

### Bring Compute Nodes Online In PBS

Set the compute nodes back to an online state in PBS.

```
qmgr -c "set node compute-node[1-4] state=free"
```

### Enable Job Scheduling

Re-enable job scheduling once everything is operational.

```
qmgc -c "set server scheduling = true"
```

### Verify Cluster Health

Verify that the PBS services are running properly and all nodes are online.

```
pbsnodes -a
```

Check if any nodes are in an offline state or have errors and correct them.

### Verify Compute Node NFS

Verify that all NFS shares are mounted properly before executing jobs,

```
pdsh -w compute-node[1-4] df -h
```

If all three NFS shares are mounted proceed to run jobs, else mount those shares with below command,

```
pdsh -w compute-node[1-4] mount -a
```

# Installing Packages in Compute Node Image

**Step 1:** Export the image path as shown below

```
#export CHROOT=/opt/ohpc/admin/images/rocky8.8
```

**Step 2:** Install the required package in master node and install the same in image using below command.

```
#dny –y --installroot $CHROOT install <package-name>
```

**Step 3:** Run the warewulf virtual node file system command

```
#wwvnfs --chroot $CHROOT
```

**Step 4:** Reboot the compute nodes to start using the installed package.

```
#pdsh –w compute-node[1-4] reboot
```

# Appendix

## C code used for matrix multiplication

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#define N 16 // Matrix size (N x N)
// Function to initialize a matrix with random values
void initialize_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            matrix[i][j] = rand() % 10; // Random values between 0 and 9
        }
    }
}
// Function to print a matrix
void print_matrix(double matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%.2f ", matrix[i][j]);
        }
        printf("\n");
    }
}
int main(int argc, char *argv[]) {
    int rank, size;
    double A[N][N], B[N][N], C[N][N]; // Matrices A, B, and C
    double local_A[N][N], local_C[N][N]; // Local parts of A and C
    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
if (N % size != 0) {
    if (rank == 0) {
fprintf(stderr, "Matrix size N must be divisible by number of processes.\n");
    }
    MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
}
int rows_per_process = N / size; // Number of rows per process
// Initialize matrices on rank 0
if (rank == 0) {
    initialize_matrix(A);
    initialize_matrix(B);
    printf("Matrix A:\n");
    print_matrix(A);
    printf("\nMatrix B:\n");
    print_matrix(B);
}
// Broadcast matrix B to all processes
MPI_Bcast(B, N * N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// Scatter rows of matrix A to all processes
 MPI_Scatter(A, rows_per_process * N, MPI_DOUBLE, local_A, rows_per_process *
                    N, MPI_DOUBLE, 0, MPI_COMM_WORLD);
// Start timing the computation
double start_time = MPI_Wtime();
// Compute the local part of the result matrix C
for (int i = 0; i < rows_per_process; i++) {
    for (int j = 0; j < N; j++) {
        local_C[i][j] = 0.0;
        for (int k = 0; k < N; k++) {
            local_C[i][j] += local_A[i][k] * B[k][j];
        }
    }
```

```
    }
// Gather the local results into the result matrix C
    MPI_Gather(local_C, rows_per_process * N, MPI_DOUBLE, C, rows_per_process * N,
                        MPI_DOUBLE, 0, MPI_COMM_WORLD);
    // End timing the computation
    double end_time = MPI_Wtime();
    // Print the result matrix on rank 0
    if (rank == 0) {
        printf("\nResult Matrix C:\n");
        print_matrix(C);
        // Print the time taken for the computation
        printf("\nTime taken: %.6f seconds\n", end_time - start_time);
    }
    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

## R code used for parallel matrix operations

```
library(pbdMPI)

# Initialize MPI
init()

# Get rank and size
rank <- comm.rank()
size <- comm.size()

n <- 5000 # Matrix size
total_chunks <- 96
chunks_per_rank <- ceiling(total_chunks / size)

# Determine local chunk range
```

```
local_chunks_start <- rank * chunks_per_rank + 1
local_chunks_end <- min((rank + 1) * chunks_per_rank, total_chunks)
local_chunks <- local_chunks_start:local_chunks_end

# Computation function
process_chunks <- function(chunks) {
  results <- matrix(0, nrow=length(chunks), ncol=13)

  for(i in seq_along(chunks)) {
    matrix1 <- matrix(rnorm(n*n), nrow=n)
    matrix2 <- matrix(rnorm(n*n), nrow=n)

    mult_result <- matrix1 %*% t(matrix2)
    rm(matrix1, matrix2)
    gc()

    result_eigen <- eigen(mult_result)$values
    rm(mult_result)
    gc()

    results[i,] <- c(
      mean(abs(result_eigen)),
      sd(abs(result_eigen)),
      quantile(abs(result_eigen), probs=seq(0,1,0.1))
    )
  }

  return(results)
}

# Start timer
if(rank == 0) start_time <- Sys.time()

# Compute local results
```

```
local_results <- process_chunks(local_chunks)

# Gather results from all ranks
all_results <- gather(local_results)

# Have rank 0 combine and save results
if(rank == 0) {
  final_results <- do.call(rbind, all_results)
  end_time <- Sys.time()
  print(paste("Time taken:", end_time - start_time))
  saveRDS(final_results, "matrix_results.rds")
}

# Finalize MPI
finalize()
```

## R code used for parallel matrix multiplication on single node

```
library(parallel)
library(doParallel)

size <- 16
cores <- 20

# Setup local cluster
cl <- makeCluster(cores)
registerDoParallel(cl)

# Create matrices
A <- matrix(runif(size^2), size)
B <- matrix(runif(size^2), size)

start <- Sys.time()
chunks <- ceiling(size/cores)
```

```
result <- foreach(i = seq(1, size, by=chunks), .combine='rbind') %dopar% {
    end_idx <- min(i + chunks - 1, size)
    A[i:end_idx, , drop=FALSE] %*% B
}
end <- Sys.time()
# Write results to file
sink("results.txt")
cat(sprintf("Cores: %d\nTime: %.2f seconds\nMatrix size: %d x %d\n\n",
            cores, as.numeric(end-start), size, size))
cat("=== Matrix A ===\n")
print(A)
cat("\n=== Matrix B ===\n")
print(B)
cat("\n=== Result Matrix (A %*% B) ===\n")
print(result)
sink()
stopCluster(cl)
```

## Job Status Codes and Their Descriptions

| Status Letter | Meaning | Description |
|---|---|---|
| **Q** | Queued | The job is waiting in the queue to be scheduled for execution. |
| **R** | Running | The job is currently running on the assigned resources. |
| **E** | Exiting | The job has finished running and is performing clean-up tasks. |
| **C** | Completed | The job has finished execution successfully or has been terminated. |
| **H** | Held | The job is held and will not be scheduled for execution until it is released. |
| **W** | Waiting | The job is waiting for a specific condition to be met (e.g., a start time). |
| **T** | Transit | The job is being moved to another server or is in the process of being staged. |
| **F** | Finished | The job has completed its lifecycle (may have finished successfully or failed). |

# Additional Resources

**HPC – High Performance Computing**

https://cloud.google.com/discover/what-is-high-performance-computing?hl=en

https://www.ibm.com/think/topics/hpc

**PBS Professional User's Guide**

https://help.altair.com/2024.1.0/PBS Professional/PBSUserGuide2024.1.pdf

**PBS Professional Administrator's Guide**

https://help.altair.com/2024.1.0/PBS Professional/PBSAdminGuide2024.1.pdf

**Linux Tutorial**

https://www.tutorialspoint.com/unix/index.htm

**R Tutorial**

https://www.w3schools.com/r

**C Tutorial**

https://www.w3schools.com/c

**Introduction to parallel computing with R**

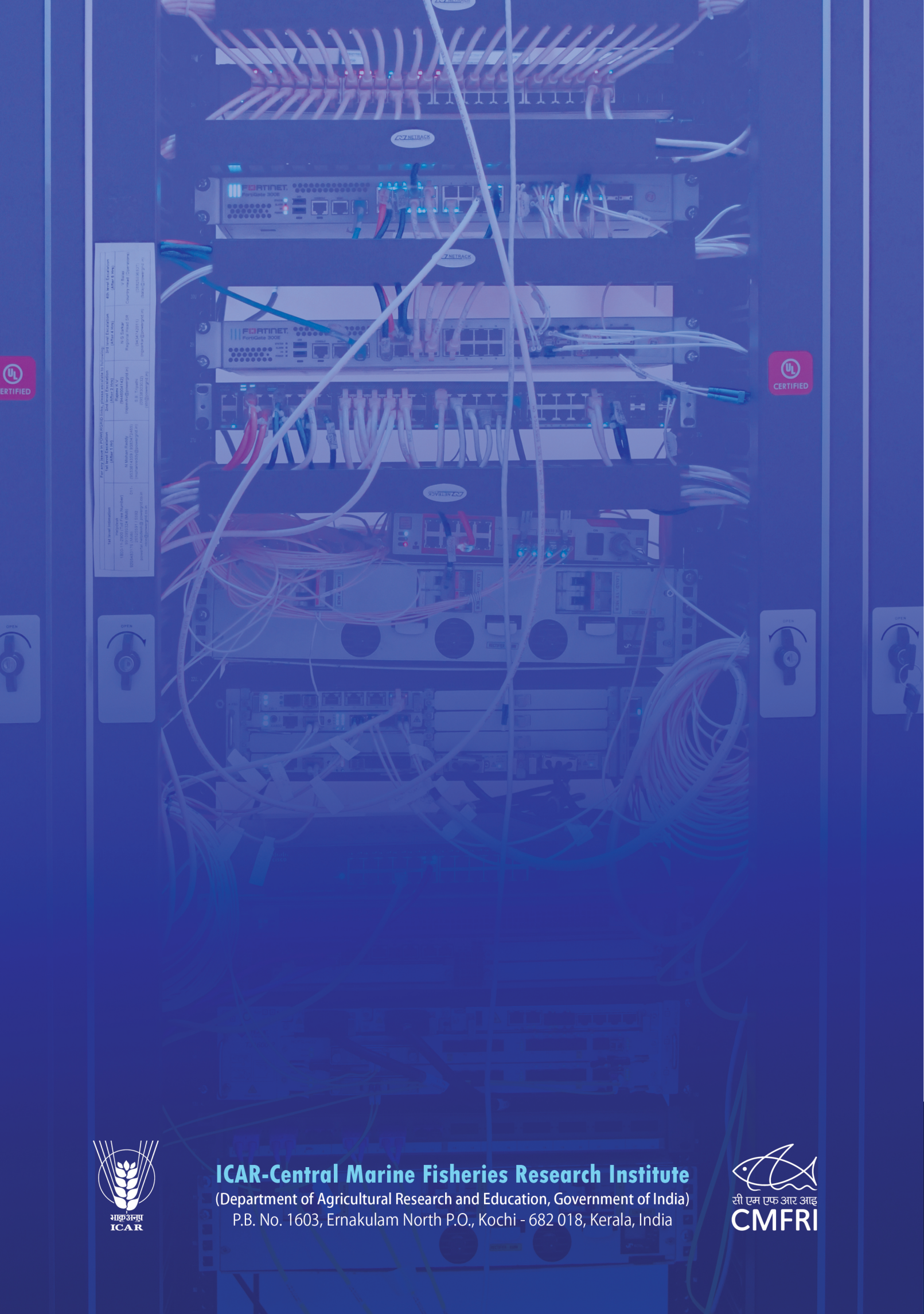
https://rawgit.com/PPgp/useR2017public/master/tutorial.html

**A Quick Guide for the pbdMPI Package**

https://cran.r-project.org/web/packages/pbdMPI/vignettes/pbdMPI-guide.pdf

**WinSCP Software**

https://winscp.net/eng/download.php




**ICAR-Central Marine Fisheries Research Institute**

(Department of Agricultural Research and Education, Government of India)

P.B. No. 1603, Ernakulam North P.O., Kochi - 682 018, Kerala, India