02_Working_in_Linux

January 12, 2020

Module 02: Working in a Linux Environment: Cluster Computer Architecture

This module is largely focussed on building an understanding of the fundamental components of cluster computer architectures and establishing a common computing framework for working on Mines computer clusters.

Many scientific/geophysical problems require computing power and storage that is beyond what is available on a single computer. In such cases, scientists often parallelize their algorithms (i.e., divide the task into smaller chunks) and run smaller jobs on multiple computers simultaneously. This module introduces the computing resources available for students at Mines and shows how students can access and utilize these resources to speed up their large-scale jobs.

0.1 What is Linux

Linux is the most popular operating system for a growing number of scientific applications, especially in geophysics. Therefore, familiarity with Linux will be advantageous throughout your career as a geophysicist, especially when dealing with large-scale numerical problems that require high-performance computing solutions.

Linux offers a free operating system. Being open-source, anybody with programming knowledge can modify it.

In addition to being a free operating system, Linux is also a great platform for programming and supports almost all programming languages people use in geophysical computing. It offers free powerful tools ranging from simple text editors to powerful integrated development environments (IDE).

title

0.1.1 Basic Linux commands

Even though many Linux distros come with a GUI, we usually interact with Linux computers through a terminal. Here, we will remember/learn some of the terminal commands for the most frequent tasks we perform on our computers: creating, copying, moving, and deleting files and folders.

0.1.2 Changing directories

If you want to change your current directory, use the **'cd'** command.

The top level directory on any Linux system is the root directory. It contains all other directories and is designated by a forward slash **'/'**. Hence, every path in Linux starts with the **'/'** symbol (e.g., /home/tugrul).

```
cd /home/tugrul
```

To navigate up one directory, use **'cd ..'**
**
Figure 1. An example Linux directory structure.
**

0.1.3 Present Working Directory

It is always useful to know where you are. You can determine the directory you are currently browsing by typing the following command

```
pwd
```

0.1.4 Listing files in a directory

To see the list of files in your current directory, use the **'ls'** command.

```
ls -l
```

0.1.5 Copying files

you can use the **'cp'** command to copy files on a Linux system. For example, **'cp /home/tugrul/main.c 'home/jeff/main_copy.c'** copies the **'main.c'** file in the **'/home/tugrul/'** directory to **'/home/jeff/'** directory and saves with the name **'main_copy.c'**.

0.1.6 Creating directories

On a Linux operating system, directories can be created with the **'mkdir directory'** command.

0.1.7 Deleting files and directories

Files and directories can be deleted using the **'rm'** command.

For example, to delete a file, you can type

```
rm filename_to_be_deleted
```

When deleting directories, you have two options. The first one is a variant of the **rm** command, but for directories

```
rmdir directory_you_want_to_delete
```

Alternatively, you can provide the **"recursive"** option to the **rm** command to delete a directory and everything inside it:

```
rm -r directory_you_want_to_delete
```

0.1.8 Moving/Renaming files and directories

Files and directories can be moved and renamed using the **'mv'** command.

To rename a directory or file, try

```
mv old_filename new_filename
```

0.1.9 Creating and using aliases

A shell alias is a shortcut to reference a command. Instead of writing long commands every time, you can create short aliases that are shorter and easier to write. The standard syntax for creating alias is given below:

```
alias shortname="your custom command here"
```

A good use of aliases for your case is when accesing the Mio cluster. Instead of typing

```
ssh -Y username@mio.mines.edu
```

each time you want to connect to Mio, you can create an alias such that

```
alias mio="ssh -Y username@mio.mines.edu"
```

In order to make aliases permanent, you need to put that into your ~/.bash_profile or ~/.bashrc file.

Another good use of aliases is to avoid typing error when using Madagascar. You can alias the common spelling mistakes for the **scons** command to ensure that it will work most of the time without requiring a correction. An example is given below:

```
# scons aliases
alias scosn="scons"
alias scson="scons"
alias csosn="scons"
alias scns="scons"
alias sconsc="scons -c"
alias sconscv="scons -c; scons view"
alias sconscs="scons -c; scons"
alias sconscv="scons -c; scons view"
```

0.1.10 top

On Linux systems, the **top** command shows the Linux processes. It provides a dynamic real-time view of the running system.

As soon as you will run this command, you will see an interactive command mode where the top half oprtion will contains the statistics of processes and the system resource usage. And the lower half shows a list of the currently running processes. To exit from this window, you just need to press the *q* button.

0.1.11 Piping in linux

A pipe is a form of redirection used on Linux-based systems to send the output of one command, program, or process to to another command, program, or process as input for further processing.You can connect programs by piping using the pipe character **"|"**. The syntax is as follows

```
command_1 | command_2 | command_3 | ... | commanc_N
```

0.1.12 grep

The grep command searches file for a particular pattern of characters, and displays all lines that contain that pattern. For instance, in a directory with so many files, you can list all the files thath contain the word "apple" by just typing the following command

```
ls -l | grep apple
```

Here, the **ls -l** command lists all the files in the current directory and sends all the listed file names to the **grep** command through pipe. The **grep** program then lists only the file names that contain the word "apple".

The **grep** command is by default case sensitive. Howeverm you can force it to make a case-insensitive search by adding the **-i** option.

0.1.13 find

You can use the **find** command find files on your Linux system. If used with the right arguments/options, find command is very powerful. The standard syntax is as follows

```
find directory_to_search -name "file_name_to_be_found"
```

You are not limited to search for files only using their names. In addition, you can also search for directories. Here are some examples on how you can search files on Linux platforms

Find all file in the entire system that are names "test.txt":

```
find / -name "test.txt"
```

Find all directories in the system called "log":

```
find / -type d -name "log"
```

Find all files in the "/usr/bin" directory of subdirectories that are larger than 1 megabyte:

```
find /usr/bin -size 1M
```

Find all files in the "/usr/bin" directory that are created more than a day ago:

```
find /use/nbin -mtime 1
```

Find all files in the "/usr/bin" directory that are created less than a day ago:

```
find /use/nbin -mtime -1
```

Find all files owned by the user "esteban"

0.2 What is a cluster?

Frequently, real-life three-dimensional geophysical problems are much larger than a single laptop or computer can handle. In such cases, we inherently need to utilize more computers to have enough computing power and memory to be able to solve the problem at hand within a reasonable time frame. Luckily, in many universities, companies, and national labs, there are large-scale computing systems that are really just collection of computers that work together to solve problems larger than any one computer can solve. These cluster of computers are often interchangably called **"clusters"**, **"supercomputers"** or resources for **"high-performance computing (HPC)"**.

The individual computers in a **cluster** are often referred to as **"nodes"**. And, just as we, people, need to talk to each other in order to collaborate, nodes in a cluster also need to be able to talk to one another; hence nodes in a cluster are connected together with a computer network (often referred to as interconnect).

Just like your laptop or desktop computers, the fundamental components of each node in a cluster are nothing but processors, memory, disks, and so on. The primary difference is the scale: nodes in clusters usually have faster processors, more memory, and more storage. And, just like your personal computer, each nodes needs an operating system to be able to function properly.

Today, computer clusters are widely used by many institutions, companies and labs. Top500,org publishes a list of the most powerful 500 supercomputers in the world along with interesting statistics twice a year. Figure 2 shows the most recent list to date that is published in November 2019.

Figure 2. The top 500 most powerful clusters in the world as of November, 2019 .

Figure 3. HPC usage by countries in terms of volume (left) and the processing power(right) as of November 2019 (TOP500).

Linux is currently the most popular operating system for clusters. Figure 4 shows the operating systems that are installed on the 500 most powerful computer systems in the world. Another good reason to learn Linux!

Figure 4. Operating systems used by the 500 most powerful computer systems in the world as of November 2019 (TOP500).

Although high-performance computers have faster processors, more memory and storage, the real power comes from using the resources in parallel than in serial. Then, the main challenge of a computational geophysicist is to rearrange the algorithms so that the large-scale problem can be divided into smaller chunks and each node can work on a smaller portion of the problem. Having a deeper understanding of the geophysical algorithms will be greatly helpful in these scenarios.

0.2.1 High-Performance Computing at Mines

Mines has a number of high-performance computing platforms: Wendian, Mio, Mc2, AuN. You can find more info on the specifications at https://hpc.mines.edu/ .

In this class, we will use Mio (see Figure 2), which has approximately 120-plus Tflop computing power. You can see the hardware on Mio below:

- 178 compute nodes
- 8-28 compute cores per node
- 2.4GHz – 3.06GHz
- 24-256 GB/Node
- Infiniband Interconnect
- 2 GPU nodes – 7.23 Tflops
- 240 TB parallel file system

Figure 3. The CSM Mio cluster.

0.3 The anatomy of a Computer Cluster

Computer clusters often include several different types of nodes, which are used for different purposes. **Master** or **Head** nodes are where you login to interact with the HPC system. master node is also the place to download/retrieve data from HPC to your local computers.

Compute nodes are where the actual heavy computing takes place. In a typical cluster setup, user usually cannot access compute nodes directly as access to these nodes are generally controlled by a scheduler.

**

Figure 4. A high-level overview of a computer cluster.

**

0.3.1 Nodes

Each node in a cluster is an individual computer with its processor and memory. Modern processors contain multiple cores that are responsible for the actual computations. In addition, there are various fast memory caches for holding the data that is currently being worked on. On the CSM Mio system, nodes have 8 or 28 cores. Often, nodes in cluster have multiple processors, which effectively doubles the number of cores per node.

Each node also has a certain amount of random access memory (RAM) available. on Mio, each compute node has 24 or 256 GB of memory.

In addition, each node has access to a disk (also called "storage" or "file system") that is usually shared among all nodes

**

Figure 4. A typical node scheme.

**

0.3.2 Storage

Usually problems people work on clusters are huge and involves many large files. In addition, these files have to be accessible from all nodes available on the system. Hence, most cluster have specialized file systems that are designed to meet these needs. Often, these file systems are intended to be used only for short- or medium-term storage. As a consequence, HPC systems may have several different file systems available, each serving for a different purpose (e.g., home and scratch file systems).

DO NOT PUT THE ONLY COPY OF YOUR SOURCE CODE TO SCRATCH!

0.3.3 The Scheduler

Computer clusters are usually shared by many users and it is common to allocate subsets of resources to different tasks. To manage the sharing of available resources amoing all of the jobs, HPC systems usually use a scheduler. Users generally, login to the master node and submit their jobs through a scheduler which runs the computation on the compute nodes. Often, schedulers decide which job to run first based on their priority that is usually assigned by users or the administrator.

0.4 Memory

The individual CPU-cores can be put together to form large parallel machines in two fundemantally different ways: the shared and distributed memory architectures.

0.4.1 Shared Memory

There are two different ways to consider regarding the terms shared memory and distributed memory: in terms of how the hardware actually implemented, and what do they mean in as programming abstractions.

From a strictly hardware point of view, we see a typical shared-memory architecture where all processors can access to a common physical memory. If you have multiple processors, then those would be able to access exactly the same memory locations (see Figure 5). This architecture is very similar to what you have in your personal computers. Hence, developing high-performance parallel algorithms for shared-memory systems is straightforward and very similar to developing serial applications that you would normally run on your laptop. However, there are two big problems with this approach: 1) building CPUs with hundred thousands of cores is not possible for now; 2) all CPU-cores compete for access to memory over a shared bus. That's this kind of systems are not manufactured today.

However, the idea of communicating directly through memory remains a useful programming abstraction. So in many systems, programmers use common programming techniques (e.g., OpenMP) to perform parallel tasks can directly access the same **"virtual"** memory regardless of where the physical memory actually exists. So, this programming model can be thought as an illusion that all memory is actually shared.

**

Figure 5. Shared memory architecture.

**

0.4.2 Distributed Memory

From hardwave point of view, in distributed memory systems, processors perform computations on their local memory and use networks to transfer data with remote processors. Although this model greatly simplifies the hardware implementation, the programming for these platforms becomes more complex as programmers also has to take of the communication between the processors in addition to actual computations.

As in the shared memory case, distributed memory can also be used as a programming model and CPU-cores can be forced to communicate through the network and behave like they are accessing their local memory even though they actually see the same common memory. So, you can use your CPU cores on your laptop as if they are on different machines.

**

Figure 6. Distributed memory architecture.

**

0.5 Connecting to the Mio System

0.5.1 Habits of a responsible HPC citizen

Before, accessing the Mio system, just bare in mind that Mio is being shared by many researchers and if you follow some basics rules, you may avoid receiving angry emails from the admin and other students or researchers.

When using the Mio system, remember the habits of a responsible HPC user.

A responsible user:

- does not use the head node for heavy calculations
- is careful about the resource usage (nodes and disk space)
- knows what to keep/delete and where to keep
- is respectful of other people's data.

0.5.2 Accessing Mio

Now that we are familiar with the high-performance computing systems and know what to expect after we access them, it is a good time to put what we learned in the previous section into practice.

Connecting to clusters is often done through a terminal using a tool known as **"SSH"** Hence, the first step of accessing Mio is opening a terminal (assuming you already have a Mio account. Otherwise, you can get an account by emailing hpcinfo@mines.edu).

Assuming you are on campus, or connected to Mines network via VPN, you can get to mo by entering the following in a terminal window:

```
ssh -Y username@mio.mines.edu
```

You will be asked for your password. The password required here is your MultiPass password.

After entering your password, you should be logged in to the **head** node and see a prompt like the one below.

[yourusername@mio001 ~]$

Since, Mio uses a Linux operating system, you can use all the commands you learned in the Linux section.

0.6 Setting up your environment

Modules Environmental variables

Because HPC systems serve many users with different software needs, HPC systems often have multiple versions of commonly used software packages installed. Since you cannot easily install and use different versions of a package at the same time without causing potential issues, HPC systems often use environment modules (often shortened to modules) that allow you to configure your software environment with the particular versions of software that you need.

On Mio, you see a list of all the available modules using the command:

```
module avail
```

Many HPC systems also have a custom environment that means that binary software package will not simply work "out-of-the-box". They may need differnt options or settings in your job script to make them work. We will see what kinds of settings we need to make in order to be able to use Magagascar on Mio.

0.7 Transferring files between local computer and Mio

To copy a file to or from Mio, you can use the **'scp'** command.

To transfer a file from your local computer to Mio:

```
scp /path/to/local/file.txt username@mio.mines.edu:/path/on/Mio
```

To download/copy a file from Mio to your local computer:

```
scp username@mio.mines.edu:/path/on/Mio/file.txt  /path/to/local/directory
```

0.8 Best advice to the programmers

Version control systems are commonly used by professional programmers. So many times, you will change something in your program and everything will stop running. Or, you may accidentally delete your source code and may have to face catastrophic consequences. To avoid all these problems, it is crucial to be aware of the availability of the version control systems and to use them regularly.

Git is the msot commonly used version control system, It tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. You have different platforms that provide such services (e.d., github, bitbucker, gitlab).

0.9 Using Madagascar on Mio

Working on the CSM Mio cluster requires developing new skills (for some!) about working in a Linux environment; writing and compiling your code in a lower-level language (e.g., C/C++/F90); submitting jobs through a job scheduler (e.g., SLURM); and debugging, validating and optimizing your code to obtain the best results in the least amount of time.

To help facilitate this process, we are going to perform our tasks within the Madagascar framework. While not a perfect environment for doing everything you might want to do in a computational sense, it does offer an extensive API that allows one to interact with (i.e., read in and write out) multi-dimensional array data stored in a regularly sampled format (RSF). There are also lots of code examples that provide a good starting point for developing your own applications.

Fortunately, there is a shared workspace in Mio, which you can access and use Madagascar by just seeting some environment variables.

To run Madagascar and have all the correct environment variables, you will need to set up your bash environment. There is a script prepared for you under **/gpfs/lb/sets/geop/M8R/"** directory.

If you have never used Mio before, you can set up your environment by typing the following commands in order.

-

```
mv ~/.bash_profile ~/bash_profile_old
```

-

```
cp /gpfs/lb/sets/geop/M8R/bashrc_generic_geop ~/.bash_profile
```

-

```
source ~/.bash_profile
```

If you are an existing user, and already have a bash setup, you can just copy and paster the contents of the **/gpfs/lb/sets/geop/M8R/bashrc_generic_geop** file to your **bash_profile** file. The most important lines

```
alias scons=/opt/python/gcc/2.7.11/bin/scons

## . . Load Mio utilities
module load utility >& /dev/null

##### Madagascar
# . . PYTHON
module load PrgEnv/python/gcc/2.7.11
# . . MPI
module load openmpi/gcc

source /gpfs/lb/sets/geop/M8R/RSF2.0/RSFSRC/env.sh

## . . Update PYTHONPATH
export PYTHONPATH=$RSFSRC/book/Recipes:/gpfs/lb/sets/geop/M8R/lib/scons-2.5.1:${RSFROOT}/lib/pyt

## . . Update PATH
export PATH=${RSFSRC}/book/Recipes/:/gpfs/lb/sets/geop/M8R/lib/scons-2.5.1/SCons:${PATH}

## . . RSF manual pages
export MANPATH=/gpfs/lb/sets/geop/M8R/RSF2.0/share/man

## . . Need to define this path
export GEOP=/gpfs/lb/sets/geop/
```

0.10 A really brief introduction to C

C is a general-purpose programming language that was mainly developed as a system programming language to write an operating system. The main features of C language include low-level access to memory, a simple set of keywords, and clean style. These features make C language suitable for system programmings like an operating system or compiler development as well as scientific programming.

The C language program can be modeled as a state machine. The state is defined by the values assigned to the variables declared in the program. The program code specifies how its state should change during program execution, which can be made by four fundamental primitives: assignment, sequential composition, branching, and loop.

Many programming languages have borrowed syntax/features directly or indirectly from C language. Like syntax of Java, PHP, JavaScript, and many other languages are mainly based on C language.

0.10.1 The structure of a C program

```
pre-processor directives

main()
{
    local variable declarations
    statement sequences
}
```

```c
/*
 * First C program that says Hello (Hello.c)
 */
#include <stdio.h>  // Needed to perform IO operations

int main() {                    // Program entry point
    printf("Hello, world!\n");  // Says Hello
    return 0;                   // Terminate main()
}
```

- simple examples
- hello world
- array example
- matrix
- functions

- LAB
- show prev code & madagascar API
- sfnoise to generate random matrix & vector
- mat-vec multiplication program
- vector norm (modular)
- run everything in mio (debug q)