GLY 6739.017S26: Computational Seismology

Notebook 03: Run C, Fortran, and Python programs on a server

Glenn Thompson | Spring 2026

This assignment introduces you to the Unix-based computational environment used throughout the course and in real scientific workflows. You will practice logging into a remote server, compiling programs, running them, and comparing performance across languages.

You are encouraged to use GenAI tools (e.g., ChatGPT) to help write code, and look up Linux command syntax as needed. The goal is understanding workflows and concepts, **not** memorizing syntax.

Important: Many steps below are *terminal commands* meant to be run on your local computer or on the remote server (`newton`). In a notebook, they are shown for copy/paste convenience.

Server Access & Unix Basics

Here we log on to the server, change your password, and make a directory.

SSH into the course server

Run this on your **local** computer in a terminal:

```
ssh [username]@newton.rc.usf.edu
```

The **ssh** program is used for secure-shell remote login.

Your username is your first name (not your USF NetID). Your temporary password is: `compsci2026!`

Note: If you are not on campus, you will need to connect to the USF Virtual Private Network (VPN). See instructions at:

https://usfjira.atlassian.net/wiki/spaces/UHID/pages/10934682250/VPN+-+Palo+Alto+GlobalProtect to download GlobalProtect and connect to the VPN.

Change your password

You might be prompted to immediately change your password. Or you can change it anytime on `newton` by using the **passwd** program:

```
passwd
```

Follow the prompts.

Create a course directory in your home directory

```
cd
mkdir -p compsci2026
cd compsci2026
pwd
```

cd changes the directory you are working in. On its own, it will change to your home directory (stored in the $HOME environment variable). But usually you follow it with the directory (or path) you want to change to.

mkdir makes a new directory.

pwd prints the working directory.

A4. Log out

```
exit
```

Checkpoint: You should be able to SSH in/out cleanly, and `~/compsci2026` should exist on `newton`.

Hello World in Three Languages
Here we write and run a minimal *Hello, world* program in three languages. We copy these onto the server to (compile and) run them, to mimic a common workflow where you write code on your local computer, and then upload it to HPC infrastructure to run it.

Create the three source files locally

On your **local** computer (not on the server), open Visual Studio Code and create these files:

- `hello.c`

- `hello.f90`
- `hello.py`

You may use ChatGPT or other GenAI tools if needed. Study the code for each. What do you notice?

Below are minimal versions you can copy/paste.

hello.c

```c
#include <stdio.h>

int main(void) {
    printf("Hello, world from C\n");
    return 0;
}
```

hello.f90

```fortran
program hello
  implicit none
  print *, "Hello, world from Fortran"
end program hello
```

hello.py

```python
print("Hello, world from Python")
```

Copy the files to `newton`

Using `scp` from your **local** terminal (adjust the path if needed):

```
scp hello.c hello.f90 hello.py
[username]@newton.rc.usf.edu:~/compsci2026/
```

scp allows you to secure copy a file (including a program) from one computer to another.

SSH to `newton` and compile/run

SSH in:

```
ssh [username]@newton.rc.usf.edu
cd ~/compsci2026
```

A **compiler** is a program that converts human-readable *source code* into machine-readable *object code* (often called an executable). Once compiled, the executable is

typically very fast to run.

Compile the C program (GNU C Compiler):

```
gcc hello.c -o hello_c
```

Compile the Fortran program (GNU Fortran compiler):

```
gfortran hello.f90 -o hello_fortran
```

List your directory:

```
ls -l
```

Notice that the *source* files usually have permissions like `-rw-rw-r--` whereas the compiled executables have execute bits like `-rwxrwxr-x`.

> **Note:** Python is typically run via the CPython interpreter (written largely in C). Python source is compiled to bytecode and executed by a virtual machine; the key point here is that your Python loop runs inside an interpreter/runtime rather than as a standalone native executable.

Run all three programs:

```
./hello_c
./hello_fortran
python3 hello.py
```

Performance Comparison on a Large Vector

In this section you will run a simple numerical workload in C, Fortran, and Python, and compare performance using `/usr/bin/time -v`.

Here is the same program, written in all 3 languages:

In C - name the file rms_large.c :

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void) {
    const int n = 10000000;  // 10 million samples
    double *signal;
    double sumsq = 0.0;
    double rms;

    // Allocate memory
    signal = (double *)malloc(n * sizeof(double));
    if (signal == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
```

```
        return 1;
    }

    // Fill signal
    for (int i = 0; i < n; i++) {
        signal[i] = sin(0.001 * (i + 1));
    }

    // Compute RMS
    for (int i = 0; i < n; i++) {
        sumsq += signal[i] * signal[i];
    }

    rms = sqrt(sumsq / n);

    printf("Samples: %d\n", n);
    printf("RMS amplitude: %.6f\n", rms);

    free(signal);
    return 0;
}
```

In Fortran90 - name the file rms_large.f90:

```
program rms_large
  implicit none

  integer, parameter :: n = 10000000
  integer :: i
  real(8), allocatable :: signal(:)
  real(8) :: sumsq, rms

  allocate(signal(n))

  ! Fill signal
  do i = 1, n
     signal(i) = sin(0.001d0 * dble(i))
  end do

  ! Compute RMS
  sumsq = 0.0d0
  do i = 1, n
     sumsq = sumsq + signal(i) * signal(i)
  end do

  rms = sqrt(sumsq / dble(n))

  print *, "Samples:", n
  print *, "RMS amplitude:", rms

  deallocate(signal)
end program rms_large
```

In Python (without numpy), name the file rms_large.py :

```python
#!/usr/bin/env python3
import math

n = 10_000_000
signal = [0.0] * n  # allocate list

# Fill signal
for i in range(n):
    signal[i] = math.sin(0.001 * (i + 1))

# Compute RMS
sumsq = 0.0
for x in signal:
    sumsq += x * x

rms = math.sqrt(sumsq / n)

print("Samples:", n)
print("RMS amplitude:", rms)
```

Copy the provided files into your directory

Create these files in Visual Studio Code, and save them. Then use **ssh** again to copy them to your compsci2026 directory on newton.

Commands (run on `newton`):

```
scp rms_large.c rms_large.f90 rms_large.py
[username]@newton.rc.usf.edu:~/compsci2026/
```

Then:

```
cd ~/compsci2026
ls -l rms_large.*
```

Compile C and Fortran with optimization

Login to newton again with **ssh** as before.

Compile the C program:

```
gcc -O2 rms_large.c -o rms_c -lm
```

Compile the Fortran program:

```
gfortran -O2 rms_large.f90 -o rms_fortran
```

Time each run

Time the compiled C executable:

```
time -v ./rms_c
```

Time the compiled Fortran executable:

```
time -v ./rms_fortran
```

Time the Python program:

```
time -v python3 rms_large.py
```

Compare these quantities across all three runs

- **User time**
- **System time**
- **Maximum resident set size** (memory)

Tip: In the `time -v` output, you can usually find lines like:

- `User time (seconds): ...`
- `System time (seconds): ...`
- `Maximum resident set size (kbytes): ...`

Python with numpy

Let's write a version of the rms_large program but with numpy - name this rms_large_numpy.py.

```
#!/usr/bin/env python3
import numpy as np
import math

n = 10_000_000

# Create vector [1, 2, 3, ..., n] and compute signal
i = np.arange(1, n + 1, dtype=np.float64)
signal = np.sin(0.001 * i)

# Compute RMS
rms = math.sqrt(np.mean(signal ** 2))

print("Samples:", n)
print("RMS amplitude:", rms)
```

Copy this to newton, and run it with the time command, as we did for rms_large.py.

Deliverables

- Compare how easy you find it to read the C, FORTRAN, Python, and "Python with numpy" programs.
- Make a table of the **User Time**, **System Time**, and **Memory Usage** for all four implementations of the `rms_large` program.
- Write 100-200 words addressing:
  - Why the runtimes differ
  - Why (pure) Python is slower
  - Why NumPy exists
  - Why C and Fortran are still used in scientific computing

Summary

Things you now know:

- how to remote login to a Linux server
- how to change your password
- how to copy files to a Linux server
- how to change directory
- how to make directories
- how to compile C and FORTRAN programs
- how to list file permissions
- interpreters are slow
- element-by-element assignment is slow (pure Python)

Notes & Expectations

- You are **not** graded on C or Fortran style.
- You are graded on:
  - successfully compiling and running programs
  - using Unix tools correctly
  - interpreting performance results
- This assignment is intentionally designed to be completed even if you have never written C or Fortran before.