Exercise 1: The Module System

Objectives:

1. Log in to the CU Research Computing System
2. Explore the CURC module stack

Estimated time to complete: 5 minutes

Connect to the CU Research Computing system using your preferred method.

Terminal (CU Boulder, CSU Users):

```
ssh <username>@login.rc.colorado.edu
```

Open OnDemand (Users from other institutions): https://ondemand-rmacc.rc.colorado.edu/ (https://ondemand-rmacc.rc.colorado.edu/)

Get on a compute node with the Alpine `acompile` program and view the Alpine module stack.

```
acompile -n 4 --time=2:00:00
module avail
```

How many compiler modules are available on Alpine?

Exercise 2: Building Software from Source

Objectives:

1. Explore CURC compilers and compiler environment variables.
2. Perform a simple source installation.

Estimated time to complete: 15 minutes

Part 1: CURC compilers and environment variables

Explore CURC compiler environment variables.

```
module load intel
module load impi
module load mkl
```

The standard compiler variables FC, CC, and CXX are set as appropriate for your compiler/MPI combination. These environment variables reference the Fortran, C, and C++ compilers respectively.

```
echo $FC
echo $CC
echo $CXX
```

In addition, several environment variables are set that may be useful during the compilation process. These variables are prefixed by `CURC` and may easily be found by searching your environment. `env | grep CURC`

These environment variables can be passed to `make` or `cmake` and make your life a lot easier by removing the need to type out long absolute paths!

Part 2: Compile a program from source.

We will be installing a common bioinformatics program, Samtools, from source. More info about the program: http://www.htslib.org/ (http://www.htslib.org/)

You will grab the Samtools source code from http://www.htslib.org/download/ (http://www.htslib.org/download/)

```
cd /projects/$USER/software
wget https://github.com/samtools/samtools/releases/download/1.17/samtools-1.17.tar.bz2
tar -xf samtools-1.17.tar.bz2
cd samtools-1.17 && ls
```

Can you tell which build system samtools requires? Which module should you load?

```
mkdir -p /projects/$USER/software/install/samtools_1.17
module load gcc
./configure --prefix=/projects/$USER/software/install/samtools_1.17
```

Are any files created or modified (or not!) during the configure step? Which one(s)? Hints: `ls -lt` `stat <file name>`

Run `make`. What do you see in `/projects/$USER/software/install/samtools_1.17`?

Finish the installation! `make install`

Was your build successful? How can you tell?

Does the following command work for you? Why or why not? Hint: Add the samtools bin to PATH! `samtools --help`

This was meant to be a learning experience, but you should always check the program's website for installation instructions! http://www.htslib.org/download/ (http://www.htslib.org/download/)

Relevant CURC Documentation

https://curc.readthedocs.io/en/latest/compute/compiling.html
(https://curc.readthedocs.io/en/latest/compute/compiling.html)

Exercise 3: Building Software with Spack

Objectives:

1. Create a Spack environment
2. Install fastqc in your Spack environment

First, load the spack module.

```
module load spack
```

Note that Spack uses the system compilers by default.

```
which gcc
gcc --version
```

Create a Spack environment. `spack env create fastqc_env`

Activate your new spack environment. Tip: You can use tab completion for spack environments! `spacktivate fastqc_env`

View the output of `spack info fastqc`.

Let's see which packages are required for a complete fastqc installation. `spack spec fastqc`

Now install fastqc v 0.11.9 **within your environment**: `spack install --add fastqc@0.11.9` Note: the installation will take ~10 minutes to complete.

Try the following commands:

```
spack env status
spack find
spack find -p fastqc
```

Notice that the last command gives you version info for fastqc and lists the location of the installation. What is in the listed directory?

Deactivate your environment with `despacktivate`

Now let's try building fastqc with a different compiler. Compilers can take a *very* long time to build, but Spack lets you add compilers that are already installed. `spack compiler add /curc/sw/install/gcc/10.3.0`

We'll need to create a new environment.

```
spack env create fastqc_gcc1030_env
spacktivate fastqc_gcc1030_env
```

Now you can install a fastqc built with gcc 10.3.0. Note that you don't have to complete this step for this tutorial. `spack install --add fastqc%gcc@10.3.0`

When the `fastqc%gcc@10.3.0` installation is complete, the output of `spack find fastqc` from **a deactivated environment** should look like this:

```
-- linux-rhel8-zen / gcc@8.5.0 ---------------------------------
fastqc@0.11.9

-- linux-rhel8-zen3 / gcc@10.3.0 -------------------------------
fastqc@0.11.9
==> 2 installed packages
```

Relevant CURC Documentation

Coming soon!

Exercise 4: Installing Software with Conda

Objectives:

1. Configure your `.condarc` file
2. Create a conda environment and install samtools
3. Run samtools from within the environment

Estimated time to complete: 15 minutes

Step 1: Configure a `.condarc` file

This step is required the first time using Anaconda/Miniconda/Mamba on CURC systems.

Navigate to your home and create a file named `.condarc`. You can use whichever text editor you are most comfortable with (`nano`, `vim`, etc.)

```
cd ~
nano .condarc
```

Enter the following text, save, and exit.

```
pkgs_dirs:
  - /projects/$USER/.conda_pkgs
envs_dirs:
  - /projects/$USER/software/anaconda/envs
```

Confirm that the text was saved. `cat .condarc`

Step 2: Create a conda environment containing `samtools`.

> ***NOTE:*** *Conda environments must be created and run from a compute (not a login) node.*`

Load the default anaconda module. `module load anaconda`

What happened to your prompt? Which anaconda environment are you in?

Create an anaconda environment and install samtools. `conda create -n samtools_env -c bioconda samtools`

Did you get an error message? Hint:

```
conda config --show channels
conda config -h
```

Enter `y` for 'yes' when asked if you want to proceed.

Step 3: Activate the environment and run samtools.

```
conda activate samtools_env
samtools --help
```

What happened to your prompt after you activated `samtools_env`?

Relevant CURC Documentation

https://curc.readthedocs.io/en/latest/software/python.html (https://curc.readthedocs.io/en/latest/software/python.html)

Useful Conda Commands (try if you have time)

```
conda env list  # list all environments
conda list      # list packages in active env
conda env remove -n <envname>   # remove an environment
conda config --show channels    # view configured channels
conda deactivate    # deactivate environment
conda create --name <clonedenv> --clone <envtoclone> # clone an environment
```

Exercise 5: Installing Software With Apptainer (formerly Singularity)

Objectives:

1. Become familiar with basic `apptainer` commands.
2. Pull an image from a pre-built container, then run the program from the container.

Estimated time to complete: 15 minutes

Basic `apptainer` commands

View a list of Apptainer commands. `apptainer --help`

Look at CURC's collection of pre-build containers.

```
echo $CURC_CONTAINER_DIR
ls $CURC_CONTAINER_DIR
```

A Singularity Definition File (or “def file” for short) is like a set of blueprints explaining how to build a custom container. It includes specifics about the base OS to build or the base container to start from, software to install, environment variables to set at runtime, files to add from the host system, and container metadata. More information from the Apptainer user guide: https://apptainer.org/docs/user/1.0/definition_files.html (https://apptainer.org/docs/user/1.0/definition_files.html)

Check out the definition file for the `mach3_build.sif` container. SIF = **S**ingularity **I**mage **F**ile `apptainer inspect --deffile $CURC_CONTAINER_DIR/mach3_build.sif`

Running programs from a container uses the following syntax: `apptainer exec <name.sif> <program command> <options>`

For example: `apptainer exec $CURC_CONTAINER_DIR/seurat_4.1.0.sif R`

Pull an image from a pre-built container, then run the program from the container.

We are going to create a containerized version of samtools using the Docker image found here:
https://hub.docker.com/r/staphb/samtools (https://hub.docker.com/r/staphb/samtools)

Note that, by default, the cache directory for Singularity builds is `/scratch/alpine/$USER`.

```
echo $APPTAINER_CACHEDIR
```

Use the `apptainer pull` command to create a `.sif` file from the Docker image. `apptainer pull samtools.sif docker://staphb/samtools`

Run samtools from the container. Is it the same version of samtools you got from conda and building from source?

Bonus question: How do `samtools_env`, `samtools.sif`, and the source installation compare in size?

Relevant CURC Documentation

https://curc.readthedocs.io/en/latest/software/Containerizationon.html
(https://curc.readthedocs.io/en/latest/software/Containerizationon.html)
