Canaan User Guide

Connecting to the Cluster

There are several different methods you can use to connect to the cluster, three of which are listed here.

SSH (Secure Shell)

The most basic way to connect to the cluster is to use SSH in a terminal application. To do this you will need to open a terminal session on your computer and use ssh to connect to the cluster. This is relatively fast to set up and is sufficient for submitting and checking on jobs, but does limit the types of editors and other tools you have access to.

Starting an ssh session from a Mac or Linux system

Open a terminal and issue the command:

```
ssh username@canaan.phys.gordon.edu
```

where `username` should be replaced by your username; normally this will be your Gordon username in the form *firstname.lastname*. Your username is case-sensitive and should be entered using lowercase letters. The first time you do this you will be met with the prompt

```
The authenticity of host 'canaan.phys.gordon.edu (10.100.49.1)' can't be established.
RSA key fingerprint is e5:76:e0:bf:61:e6:41:eb:6e:8a:df:aa:81:a8:19:58.
Are you sure you want to continue connecting (yes/no)?
```

Type `yes` and press `Enter`. You will then be prompted for your password.

Starting an ssh session from a Windows PC

You will need to use Putty or some other terminal emulation program that supports SSH. Putty is easy to install; just download putty.exe and save it to your desktop. Start Putty then do the following steps (These need be done only once; from now on when you start Putty you can click on the saved session name then click on "Open" to initiate the connection):

1. Enter **`canaan.phys.gordon.edu`** in the field labeled "Host Name (or IP address)"
2. Enter **Canaan** (or whatever you want to name this connection) in the field labeled "Saved Sessions"
3. Click the "Save" button on the right side of the window
4. Click the "Open" button at the bottom of the window

Once you're connected...

- Log in using your username (if not already entered) and password. *These are case-sensitive!*

- You have limited editor choices when working in a terminal through an ssh connection. These include *nano* (the best choice if you are not already familiar with unix/linux editors), *emacs*, and *vi*. GUI editors will not normally be available.

Ending an ssh session

When you're done, type `exit` at a command prompt and press `Enter`. You can also type `logout` and press `Enter` or just press *Ctrl-D*. Any of these will end your terminal session.

Configuring SSH for login without typing a password

Normally when you connect via SSH you will be prompted for your password. It is possible to configure your account on Canaan to accept connection requests from your account on another machine without requiring you to type in a password. This is done through an SSH key pair. The following instructions cover how to set this up on a Mac or Linux machine. (See https://winscp.net/eng/docs/ui_puttygen for instructions on how to do this with Putty.)

Log in to your computer and start a terminal. At the command prompt type

```
ssh-keygen
```

Assuming you have not yet done this, you will see several prompts; you can press `Enter` in response to each of them. The resulting output will look something like

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jane.doe/.ssh/id_rsa):
Created directory '/home/jane.doe/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jane.doe/.ssh/id_rsa.
Your public key has been saved in /home/jane.doe/.ssh/id_rsa.pub.
The key fingerprint is:
39:1d:3f:f6:a4:7f:9a:f2:df:2d:d1:20:0d:cf:a3:ba jane.doe@some_machine
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|            .    |
|          .  =   |
|         o o. *  |
|        S . +o.+ |
|         . ..=. .|
|           .. .. |
|          . ....+|
|          E. o==+|
+-----------------+
```

This command has created a "key pair" and placed them in the subdirectory (folder) `.ssh`. Change to this directory and list the files there by typing

```
cd ~/.ssh
ls -l
```

at the command prompt in the terminal. The output will look something like

```
total 8
-rw------- 1 jane.doe  jane.doe  1679 Nov 12 08:09 id_rsa
-rw-r--r-- 1 jane.doe  jane.doe   398 Nov 12 08:09 id_rsa.pub
-rw-r--r-- 1 jane.doe  jane.doe   884 Nov 12 07:49 known_hosts
```

The file `id_rsa` contains your private key and should **never** be shared or copied anywhere (except to a secure backup location). The file `id_rsa.pub` contains your public key and can be installed on remote computers like Canaan and so that they will allow login attempts from your computer without a password. The file `known_hosts` contains an identifier for each of the computers you have already made SSH connections to; an entry will be added to this file each time you connect to a new host.

Since your username on your machine is probably not the same as the one you use on Canaan, we'll first configure SSH so it automatically will use the correct username. You will need to use a text editor on your computer to create a file in the `.ssh` directory; the instructions here offer one way to do this but you can use any method you like. Make sure you are in the `.ssh` directory and use an editor of your choice (`nano` works well if you don't already have a favorite) to create a new file named `config`:

```
cd ~/.ssh
nano config
```

Type in the following, replacing *<username>* with your username on Canaan

```
host canaan.phys.gordon.edu
    user <username>
```

Notice at the bottom of the window there are two lines with some basic keystrokes. Type *Ctrl-X* to exit the file, respond with **y** when asked to confirm writing the file, and press `Enter` to accept the default name (`config`).

Execute the following commands. You will probably be asked for your password several times; use the password for your account on Canaan:

```
ssh canaan.phys.gordon.edu mkdir .ssh
ssh canaan.phys.gordon.edu chmod go-rwx .ssh
scp id_rsa.pub canaan.phys.gordon.edu:.ssh/authorized_keys
ssh canaan.phys.gordon.edu chmod go-rwx .ssh/authorized_keys
```

You're done! Try connection to Canaan with

```
ssh canaan.phys.gordon.edu
```

and you should be logged into your account right away. This also means that you can easily copy files back and forth from Canaan using `scp` or `sftp` without having to type your password. Type

```
man scp
man sftp
```

for more information on how to use `scp` and `sftp`.

X2Go

X2Go is a Remote Desktop solution that enables you to access the graphical desktop of a computer over a low or high bandwidth connection. This is the recommended way to connect to the cluster as it gives you access to the full range of applications, including GUI editors and cluster monitors.

One-time setup

- Browse to http://wiki.x2go.org/doku.php/doc:installation:x2goclient and download an X2go client for your system. Clients are available for Windows, OS-X, and many distributions of Linux.
- Install the client according to instructions on the client download page.
- Start the X2go client and click on the "New Session" icon in the upper left corner and enter in the following information:
    - Session name: **Canaan** (although you can name this anything you want)
    - Host: **canaan.phys.gordon.edu**
    - Login: **<your username on Canaan>** (this is *case sensitive*)
    - Optional - If you've configured SSH to allow password-less login, click the checkbox for **Try auto login (ssh-agent or default ssh key)**
    - Session type: **GNOME** *(select using the pull-down menu)*
- Click "OK" to create the session launcher. The new session launcher will appear on the right side of the client window.

Starting an X2Go session

Configured X2Go sessions will appear on the right side of the X2Go window.

- Before connecting, select the desired window size from the pull-down menu for the session. The default is 800x600 but you may prefer to use a larger size such as 1280x1024.
- Click on the session name to start a session. If you are prompted for your password, type it in (remember that it is case-sensitive). After entering it a new virtual desktop window should appear and you should be logged into Canaan.
- If an authentication window appears asking you "Password for root:" you can click "cancel" to dismiss the window.
- Much of the work on Canaan is done using commands typed into a terminal window. To open a terminal window, use **Applications** -> **System Tools** -> **Terminal**. (You can drag the terminal icon from the **System Tools** menu to the panel at the top of the desktop; this allows you to open a terminal window with a single click.)

Ending an X2Go session

- To end your session, click on your name in the upper right of the virtual desktop, select **Quit...**, and then click on **Log Out**. **Important:** *Please be sure to do this otherwise your session will remain active.*

Remote Desktop

If you have a Remote Desktop client installed on your machine you can use it to connect to the cluster.

Starting a remote desktop session

- Enter **canaan.phys.gordon.edu** for the hostname and log in using your assigned username and password (don't forget these are case-sensitive). After connecting a new virtual desktop window should appear and you should be logged into Canaan.
- If an authentication window appears asking you "Password for root:" you can click "cancel" to dismiss the window.
- Much of the work on *Canaan* is done using commands typed into a terminal window. To open a terminal window, use **Applications** -> **System Tools** -> **Terminal**. (You can drag the terminal icon from the **System Tools** menu to the panel at the top of the desktop; this allows you to open a terminal window with a single click.)

Ending a remote desktop session

- To end your session, click on your name in the upper right of the virtual desktop, select **Quit...**, and then click on **Log Out**. **Important:** *Please be sure to do this otherwise your session will remain active.*

Using a Linux System

The *Canaan* cluster uses CentOS 6 Enterprise Linux.

Note: Commands in Linux are case-sensitive; you will rarely, if ever, have to type a command with upper-case letters.

One of the first things you should do is **change your password**. If you connected using remote desktop or X2Go, you will need to open a terminal shell. Do this by clicking on **Applications** -> **System Tools** -> **Terminal**. If you connected via SSH then you are already using a terminal shell. Type `yppasswd` at the command prompt and press `Enter`. You will be prompted for your current password then asked for a new password and verification. *Do not use the `passwd` command as it will not work.*

Some resources that you might find helpful if you are new to working with Linux on *Canaan* include:

- LinuxCommand.org
- Chapters 8 through 18 (especially chapter 16) of CentOS 6 Essentials

Many different commands are introduced in the sections below. You can find usage information and many other details about most of them using the man command. For example, the `ls` command is used to list the contents of a directory (folder). To find out about all the options this command has type

```
man ls
```

Use the spacebar to advance through the manual page one screen at a time. Press **q** to quit reading and return to the prompt.

Using Environment Modules

Overview

It is often desirable in HPC work that different versions or implementations of compilers, libraries, or other packages be present on a machine or cluster. The Lmod Environment Modules System makes it easy to choose between them. It is important to note that

Environment Modules are used mainly for locally installed software. System software, such as the system version of Python, can be used without having to do anything with modules.

You can see what modules are currently loaded by typing

```
module list
```

Assuming only a few default modules are currently loaded, will resemble

```
Currently Loaded Modules:
  1) gcc/native   2) smake/1.5   3) StdEnv
```

You can see what modules are available by typing

```
module avail
```

The output indicates the names and version numbers of the modules that are currently available. To load a module use the `module load` command. For example, to load the `openmpi` module one would type:

```
module load openmpi
```

If you do this and then type the `module avail` command again you should see a few more lines; these new lines represent modules that are only available when the `openmpi` module is loaded.

Modules are hierarchical and the hierarchy is read from bottom to top in the output of the module avail command. The modules `atlas`, `fftw`, `mpich`, and `openmpi` all depend on the GNU C/C++ compiler and libraries being present, and so these modules are all available because the `gcc` ("native" or "system version") module is already loaded. Likewise, since the `openmpi` module is loaded, the module `hdf5` that is compatible with OpenMPI is available.

Now try switching the `openmpi` module with the `mpich` module:

```
module load mpich
```

and use the `module avail` command again. It will look similar, except that you will see that the `hdf5` module is now located in `/shared/modulefiles/MPI/gcc/native/mpich/<version number>` rather than in `/shared/modulefiles/MPI/gcc/native/openmpi/<version number>`. Conveniently, if the `hdf5` module had already been loaded, swapping MPI implementations would automatically swap `hdf5` modules so that the proper one will be used.

Module command summary

To get help with the module command type

```
module help
```

Here is a list of some useful module commands:

`module avail`	display available modules
`module help`	display help information for the module command
`module help <module>`	display help for specified module
`module list`	list currently installed modules
`module load <module>`	load specified module
`module purge`	unload all modules
`module swap <old module> <new module>`	replace currently loaded module with another module
`module unload <module>`	unload specified module
`module whatis <module>`	display one-line description of specified module

Setting default modules

It is convenient to create a default set of modules that will be loaded automatically when you start an interactive session. Once you've loaded the modules you want available by default, type

```
module save
```

to save the currently loaded set of modules as your default set.

More information on Environment Modules

For more information on Lmod Environment Modules, including several user guides, please see https://www.tacc.utexas.edu/tacc-projects/lmod.

Running Jobs on the Cluster with SLURM

Cluster Status

Jobs are run on the cluster through a workload manager or resource manager, which is responsible for allocating the cluster's resources to jobs. Submitted jobs are placed on a queue and the *workload manager* (also called a *job scheduler*) assigns them to a processor or processors as they become available. On Canaan we are using SLURM (Simple Linux Utility for Resource Management) as the workload manager.

The command

```
sinfo
```

will list all cluster nodes along with their state, often either `idle`, `idle~`, `mix`, or `alloc`. The state `idle` means the node is powered on and available and all of its CPU cores are currently free and can be allocated to a job. The state `idle~` indicates the node is currently powered off but will be automatically started when it is needed. The states `mix` and `alloc` indicate that some or all of the node's cores are allocated to jobs.

The command

```
squeue
```

shows a list of jobs currently running on the cluster, including the job name, the user who submitted the job, the job's current running time, and the node(s) the job is running on. The command

```
sview
```

opens a window that provides the same information as squeue but with a graphical user interface (this will work if you're using remote desktop or X2go; it will only work via ssh connections in very specific circumstances).

Running Jobs Interactively from the Command Line

Running Sequential (single thread) Programs

Programs can be run on the cluster using the `srun`. To run the executable program `my_prog` on one of the cluster nodes you would type

```
srun ./my_prog
```

If the cluster has any free processors (CPU cores not currently assigned to a job), the program will be run immediately. If all processors are in use then the program will queued and will run as soon the necessary resources become available.

A Python program called `my_prog.py` can be run with

```
srun python ./my_prog.py
```

It is possible to run multiple instances of a program by including the `-n` (or `--ntasks`) switch with the `srun` command. For example, to run 10 instances of `my_prog` one would type

```
srun -n 10 ./my_prog
```

or

```
srun --ntasks=10 ./my_prog
```

Running Multithreaded Programs

It is possible to run a multithreaded program on a cluster node to take advantage of the 8, 12, or 16 cores each node has. There are several ways to do this, but one simple way is to provide the `--exclusive` switch to `srun`; this indicates that the submitted job will be the only one that SLURM schedules to run on a particular node. For example:

```
srun --ntasks=1 --exclusive ./my_multithreaded_prog
```

Running Batch Jobs

Although less convenient, additional flexibility is possible by using SLURM as a batch scheduler. A small job-script program must be written to specify job parameters such as job name, required resources, expected run time, and perhaps an instruction to send the user an email when the job completes. These parameters are used by the scheduler to ensure that all necessary resources will be dedicated to the job when it runs. The job script also includes the commands necessary to complete the job; usually this is a command to run a single program but any number of shell commands can be used.

The job scheduler on Canaan is SLURM. A very simple SLURM job script to run a single instance of the Python program `my_prog.py` might look like

```
#!/bin/bash
#SBATCH --job-name=my_prog
#SBATCH --ntasks=1
#SBATCH --time=00:20:00
#SBATCH --output=my_prog-%j.out
#SBATCH --error=my_prog-%j.err
srun python ./my_prog.py
```

The first line is required and indicates the shell interpreter that will be used to process the job script. Each line beginning with `#SBATCH` sets one or more SLURM parameters. In this example the parameters set are

1. the job name
2. the number of tasks (parallel processes)
3. the maximum time for the job in HH:MM:SS format
4. the name of the standard output file (`%j` is replace by a unique job number)
5. the name of the standard error file

The `sbatch` command is used to submit a batch job. For example, suppose the batch job file is named `my_prog.sh`. The "`.sh`" extension indicates the file is a bash shell script file. The extension can be anything as long as the first line in the file is `#!/bin/bash`. Submitting the batch job is done with

```
sbatch my_prog.sh
```

If all the necessary resources are available the job will start running and you will see a message indicating the job has been submitted. If needed resources are allocated to other jobs you will see a message indicating the job has been queued until resources become available.

Suppose you want to run many copies of this program all at the same time. Only two changes are necessary:

1. set the desired number of tasks using the `--ntasks` directive
2. use the `srun` command when running the program:

```
#!/bin/bash
#SBATCH --job-name=my_prog_parallel
#SBATCH --ntasks=24
#SBATCH --time=00:20:00
#SBATCH --output=my_prog_parallel-%j.out
#SBATCH --error=my_prog_parallel-%j.err
srun python ./my_prog.py
```

In this case 24 instances of the program will be run.

Note that it is possible to override #`SBATCH` directives in the batch file on the sbatch command line. The command

```
sbatch --ntasks=24 my_prog.sh
```

will cause the job to use 24 tasks regardless of the number of tasks requested in the file.

Batch Job-Script Template

Job-script program names should have the suffix "`.sh`" or "`.sbatch`" and should not contain any whitespace (i.e., do not use spaces in the file name). It is often helpful to add some extra lines to the job-script to provide additional information about the job (time started, time finished, working directory, etc.) Below is a template that you can use cut-and-paste to create a job-script with some of these features. To use it, just replace the text bracketed by the `<FIXME>` and `</FIXME>` tags. Refer to the `sbatch` manual page for more information about the various directives.

```
#!/bin/bash
### SLURM/SBATCH bash script
###
### ***************************************************************************
### * NOTE: The lines that begin with #SBATCH ("pound-SBATCH") are batch    *
### *       scheduler directives, so they are absolutely crucial.           *
### *       DON'T REMOVE THE pound sign (#) from before the SBATCH!!!!      *
### *       Any additional pound signs in front of SBATCH will disable      *
### *       the scheduler directive.                                        *
### *                                                                       *
### * NOTE: If you create any file of human-readable text on a Windows      *
### *       machine, you *MUST* perform the following command on it:        *
### *                                                                       *
### *           dos2unix <filename>                                         *
### *                                                                       *
### *       This is because virtually all text editors in Windows embed     *
### *       hidden special characters in text files and these hidden        *
### *       special characters can cause Unix/Linux programs to fail.       *
### ***************************************************************************
###
### Submit a batch job:  sbatch <script_name>
###
### See queued jobs:     squeue
###
### See job history
###  - current day:      sacct
###  - from date:        sacct -S 2017-01-01
###  - all users:        sacct -a
###  - including nodes:  sacct -o jobid,jobname,user,state,alloccpus,nodelist
###
### Cancel a batch job:  scancel <jobid>
###
### -------------------------------------------------------------------------

### Set the job name
#SBATCH --job-name=<FIXME>job_name (no spaces in name)</FIXME>

### Define output file names
#SBATCH --output=<FIXME>job_name</FIXME>-%j.out
```

```
#SBATCH  --error=<FIXME>job_name</FIXME>-%j.err

### Specify the number of CPUs for this job:
### to run N tasks on any available CPU cores, use
###   #SBATCH --ntasks=N
###
### to run N tasks, each using M CPU cores, use
###   #SBATCH --ntasks=N
###   #SBATCH --cpus-per-task=M
###
### to run N tasks, with at most M tasks on a node, use
###   #SBATCH --ntasks=N
###   #SBATCH --ntasks-per-node=M
###
### to run N tasks, each with exclusive access to the GPU, use
###   #SBATCH --ntasks=N
###   #SBATCH --gres=gpu
###
#SBATCH --ntasks=1

### Specify the anticipated run-time for this job using HH:MM:SS format
#SBATCH --time=00:05:00

### Send email when job aborts or completes
#SBATCH --mail-type=END
#SBATCH --mail-type=FAIL
#SBATCH --mail-user=<FIXME>firstname.lastname@gordon.edu</FIXME>

### Display the job context
echo "************************************************************************"
echo "* JOB: ${SLURM_JOB_NAME}"
echo "* JOB ID: ${SLURM_JOB_ID}"
echo "* Directory is $(pwd)"
echo "* Using ${SLURM_NTASKS} task$(((SLURM_NTASKS>1)) && echo s) across\
 ${SLURM_JOB_NUM_NODES} node$(((SLURM_JOB_NUM_NODES>1)) && echo s) with\
 ${SLURM_CPUS_PER_TASK:-1} CPU core$(((SLURM_CPUS_PER_TASK>1)) && echo s)\
 per task"
echo "* Start time is $(date)"
echo "************************************************************************"
echo

### Run the job.
srun <FIXME>executable command</FIXME>

### All done, finish up.
echo
echo "************************************************************************"
echo "* End time is $(date)"
echo "************************************************************************"
```

Stopping Jobs

Jobs can be killed and removed from the system using the `scancel` command. Every SLURM job has a job ID which is reported by `srun` or `sbatch` when the job is started. You can also find the job ID by running the `squeue` command. Run `scancel` with a job's ID to cancel it. For example, if the job ID is 2378, one would type

```
scancel 2378
```

Using MPI For Parallel Programming

Selecting an MPI version to use

There are two different implementations of MPI available on this system, OpenMPI and MPICH, both of which conform to the MPI standard. You must choose one of them and load it using the module load command before compiling or running an MPI program. As of January 2016 **you are advised to use OpenMPI** as it uses an InfiniBand network to connect nodes; this is *much faster* than the TCP/Ethernet used by our installation of MPICH.

To select OpenMPI, type

```
module load openmpi
```

To select MPICH, type

```
module load mpich
```

To change from one implementation to another you only need to load the version you want to use; the other will be replaced automatically.

You can determine which implementation is currently loaded by typing `module list`.

Compiling MPI programs

Once an MPI module is loaded you can compile MPI programs. The following compiler front-ends are available with all installed MPI distributions:

- `mpicc`: C compiler
- `mpic++`: C++ compiler (also could use `mpicxx`)
- `mpif77`: Fortran 77 compiler
- `mpif90`: Fortran 90 compiler

The front-ends take care of locating the `mpi.h` header and linking the MPI libraries when an executable is built. For the most part you can use them just as you would one of the GNU compilers. For example, a C++ program can be compiled with

```
mpic++ -O2 -o myprog myprog.cc
```

Running MPI Programs

Note: As of January 2016 it is **highly recommended that you use OpenMPI** rather than MPICH since it uses the InfiniBand network (much faster than 1 Gig Ethernet).

The SLURM command `srun` can be used to run MPI-based programs on the cluster. A typical usage is

```
srun -n 48 ./my_mpi_prog
```

(remember that the `-n` flag is short for `--ntasks`). This allocates 48 cores and runs `my_mpi_prog` using these cores. Another example is

```
srun --ntasks=48 --ntasks-per-node=4 --exclusive my_mpi_prog
```

This does the same thing except it limits the number of jobs assigned to each node to 4 and requires that no other jobs are running on the node. Please read the `srun` man page for more information on the options available to this command.

A simple `sbatch` script to run the same program might look like

```
#!/bin/bash
#SBATCH --job-name=my_mpi_prog
#SBATCH --ntasks=48
#SBATCH --ntasks-per-node=4
#SBATCH --exclusive
#SBATCH --time=00:20:00
#SBATCH --output=my_mpi_prog-%j.out
#SBATCH --error=my_mpi_prog-%j.err
mpiexec ./my_mpi_prog
```

See the Batch Job-Script Template section above for more information.

Monitoring with Ganglia

Ganglia is a web-based monitor system with a graphical user interface. Browse to http://canaan.phys.gordon.edu/ganglia/?c=Canaan&m=load_one&r=hour&s=by%20name&hc=3&mc=2 to see the current status and recent history of the Canaan cluster.