A quick reference guide for working with the brook cluster. Code prefaced by a `$`-sign is to be interpreted as a shell command. When entering a command the `$` is not to be retyped into the terminal. Text inside `<...>` has to be modified.

0.1 Brief System Description

- Head node: `csg-headnode`
- 2 compute nodes to be used for the lecture: csg-brook01 and csg-brook02
  - CPU: AMD EPYC 7351P
  - RAM: 256 GB
  - GPU: 8x Nvidia RTX 2080Ti
- Interconnect: Gigabit Ethernet

1 ssh

1.1 Basic Login

The easiest way (not necessarily the most convenient) to connect to a remote machine is via invocation of the basic `ssh` command from the terminal. Linux, Mac, and BSD users should have `ssh` installed by default. Windows users can use the Windows Subsystem for Linux (WSL) to open a bash prompt or alternatively use the program PuTTY[1]. `ssh` expects a username and a hostname separated by an `@` symbol.

```
$ ssh emlXX@csg-headnode.ziti.uni-heidelberg.de
```

After successfully entering your password you should be greeted by a command prompt. To close the session simply type `exit`, `Ctrl-D`, or just close your terminal emulator.
To update your password you can use:

```
$ passwd
```

This will prompt you for your initial password and let you set a new one. Make sure to share the new password with your group mates!
Always use the head node `csg-headnode` to log into the system. Also, perform typical auxiliary tasks like compilation on this machine. The other nodes: *csg-brook{01,02}*, *csg-octane{02..08}* are reserved for compute tasks.

1.2 (Optional) Configuration File

Retyping the above command can become tedious over time. By using an ssh configuration file this can be shortened, however. Under Linux, Mac, and BSD you can edit (or create if it not yet exists) a config file for ssh under `$HOME/.ssh/config`. The file follows a simple syntax (see here for extensive information). A Basic example is listed below:

[1] https://www.chiark.greenend.org.uk/~sgtatham/putty/

Listing 1: Basic ssh configuration file

```
Host headnode
        HostName csg-headnode.ziti.uni-heidelberg.de
        User emlXX
```

Now you can simply use:

```
$ ssh headnode
```

You will still be prompted for a password though. PuTTY users can just save the connection via the GUI.

1.3 (Optional for shell users) Login Without Password

To avoid having to reenter your password every time you log on, you can use an ssh-key for authentication. First create a new ssh-key pair via:

```
$ ssh-keygen -t ed25519
```

This will prompt you for a filename. You can call the key whatever you like. I would suggest calling it something like `eml-lecture` for easy identification. This will create two files: a private key `<your-key-file>` and a public key `<your-key-file>.pub` Next, copy the key to the server's `authorized_keys` via:

```
$ ssh-copy-id -i <your-key-file> emlXX@csg-headnode.ziti.uni-
  ↪ heidelberg.de
```

You can tell ssh to use your key automatically by editing your ssh-config:

Listing 2: ssh config with automatic login

```
Host headnode
        HostName csg-headnode.ziti.uni-heidelberg.de
        User emlXX
        IdentityFile <path-to-your-keyfile>
```

You should now be able to automatically log in via:

```
$ ssh headnode
```

2 Slurm

You don't have direct access to the cluster's compute nodes. Instead, Slurm is used as a workload manager. Slurm handles your jobs and distributes them among the compute-nodes according to your specification. Slurm takes care of allocating resources and scheduling jobs. For more information on Slurm see here.
There are several so-called partitions e.g. `all, brook, octane, exercise_eml, exercise_hpdc, exercise_gpu` You only have access to the exercise partition and compute nodes `csg-brook{01,02}`. There is also a default job time limit of 12h.
Use the `sinfo` command to see the available nodes and their status.

2.1 (Basic) Running Jobs

You can start jobs by using the `srun` command:

```
$ srun -p <partition> -w <node> --pty -- <command>
```

Where `w` accepts a single node or a comma separated list of node names.
Tip: use `sinfo` and `squeue` to see if a node is currently blocked by another group.

2.2 Consumable Resources

Resources like GPUs and CPU cores can be requested using the `--gres` option. Check the message of the day when logging in for more details.
The following will command will allocate a GPU.

```
srun --gres=gpu:1 <...>
```

More CPU cores can also be requested.

```
srun --cpus-per-task 8 <...>
```

2.3 Canceling Jobs

If you want to terminate a job, e.g. due to a deadlock or other error, use the `squeue` command to get its job ID. To see only jobs started by you provide your username with the `u` option:

```
$ squeue -u <username>
```

Once you have identified the job's ID use:

```
$ scancel <jobid>
```

to stop the job.

2.4 (Advanced) Running Jobs

Slurm supports special shell scripts called sbatch files. These work like normal shell scripts but may contain special instructions for Slurm that are prefaced by `#SBATCH`. The corresponding command is `sbatch`:

```
$ sbatch <batch-file>
```

This is especially handy for starting multiple jobs with different launch parameters with one command. See also this very good summary of how to use `srun` and `sbatch`.

2.5 Conda – Software Environment Management

We use *conda* to manage different versions of software available. Use

```
$ conda activate eml
```

to enable the default PyTorch environment used for the lecture.
You may be prompted on first use to run

```
$ conda init <bash|zsh>
```

depending on your login shell. This will modify your `.bashrc/.zshrc` file to load the appropriate paths and will change your prompt to show the currently loaded environment.

```
(base) csg-headnode% conda activate eml
(eml) csg-headnode%
```

3 Editing and File Transfer

There are multiple ways to get your programs and results to and from the cluster.

3.1 Editing Locally

You can edit your files on the head-node directly from the terminal. The editors `(neo)vim` and `nano` are preinstalled. If you are unfamiliar with `vim`, `nano` should be more straight forward.

```
$ nano <file>
```

3.2 scp

`scp` is both the name of the protocol and program used for copying files between remote machines using ssh. It works exactly like its local counterpart `cp`. If you want to copy a file from your local pc to the cluster use:

```
$ scp <file> headnode:<path-to-destination>
```

This also works in the other direction. If you want to copy a file back from the headnode use:

```
$ scp headnode:<path-to-file> <file>
```

NOTE: if you want to copy a folder do not forget to use the `-r` flag to recursively copy subdirectories and files.

3.3 rsync

A more sophisticated option to copy files is `rsync`. `rsync` not only copies files, but also keeps track of changes like a version control system. It will not re-copy edited files but simply send a diff to the remote host which will update the file accordingly. Please read the documentation for usage. A basic example:

```
$ rsync -r --update <folder> headnode:<path-to-destination>
```

3.4 sshfs

You can also mount your development folder directly to your file system using `sshfs`. This has the added benefit that you can edit all of your files directly in your favorite editor or IDE on your local machine. Depending on your network connection this can be flaky at times. You may want to pass the `-o reconnect` option to automatically reconnect if the connection is lost.
First create a mount-point for the remote file system, e.g.:

```
$ mkdir mnt-remote
```

Then mount the remote file system:

```
$ sshfs headnode:<path-to-folder> mnt-remote
```

Again consult the linked wiki article for more information on the options etc.

3.5 Windows

Windows users can follow this tutorial on how to use ssh and scp from the Windows Powershell.