Introduction to Linux & RCC

Igor Yakushin
ivy2@uchicago.edu

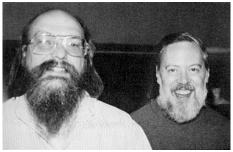
June 24, 2019

Where to get this tutorial

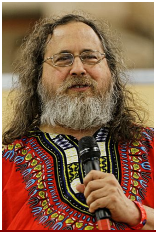
- Point your browser to
  `https://git.rcc.uchicago.edu/ivy2/Linux`

History

- UNIX operating system, the commercial ancestor of Linux, was developed by Ken Thompson and Dennis Ritchie at AT&T Bell Labs in 1969 and released in 1970



- Later several more commercial UNIX distributions appeared
- In 1983, Richard Stallman started the GNU project with the goal of creating a free UNIX-like operating system.



History

- By the early 1990s, there was almost enough available software to create a full operating system. However, the GNU kernel did not work out.
- In 1991, while studying computer science at University of Helsinki, Linus Torvalds, at the age of 21, began a project that later became the Linux kernel

- GNU/Linux provided free open source UNIX environment without expensive license fees
- Nowadays GNU/Linux is usually just called Linux

History

- It is free, open source and a lot of developers and companies from all around the world keep contributing to Linux kernel and application software
- When I started using Linux in 1994, it felt almost illegal:
  - your employer would typically be very suspicious if you are not using the blessed Windows
  - no vendors would talk to you if you say you need their hardware to behave under Linux
  - due to no vendor support one had to be very careful when buying hardware to make sure that all the components would work with Linux
  - it was quite an adventure to install it
  - many useful applications were not available under Linux and one had to use Windows for those
- These days Linux is the mainstream server platform, it is used on almost all HPC sites, embedded devices, etc.
- It is used by such companies as Google, Facebook, Oracle, banks from Wall Street, etc.

History

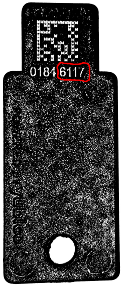
- Even Microsoft these days relies on Linux in their Azure cloud platform and is one of the major contributors to Linux
- There is a good vendor support now, especially for servers and HPC systems, although still not necessarily for consumer laptops/desktops
- There are free and commercial applications to do anything under Linux and since the beginning of the century I use Linux for everything and have no use for Windows
- It is easy to install Linux today
- There are hundreds distributions to select from: Debian, RedHat, Scientific Linux, Fedora, Slackware, Gentoo, etc.
- Two big families of distributions: Debian-based and RedHat-based
- For a typical user it is recommended to start with one of the most popular distributions: for example, Ubuntu or CentOS - huge user community, easy to find answers to your questions, more free and commercial software works out of the box, more frequent updates, security patches are available

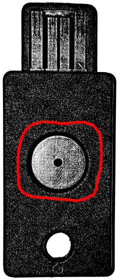
History

- If you want to install Linux and start experimenting with it, the easiest thing to do is to install VirtualBox (https://www.virtualbox.org) which is available for all the major OSes (Windows, MacOS, Linux) and install Linux as a virtual machine before you are ready to put it on bare hardware
- That way you can quickly experiment with various Linux distributions without having to wipe out the existing OS
- 99% of this talk applies to Mac as well even though one can completely avoid command line by using GUI. MacOS is also based on UNIX and its command line is not that much different from Linux.
- Strictly speaking you can get away today with using Linux via GUI and not dealing with command line, at least on your desktop/laptop but not on HPC clusters. However, using command line, once you get used to it, is actually easier and gives you more power so it is worth learning it.

Login to midway: yubikey

- If you do not yet have a midway account, I can provide a guest account using yubikey
- However, notice: it does not work with ThinLinc or Jupyter, only ssh





- Use the last 4 digits of yubikey `<XXXX>` as part of userid:
  `ssh -Y rccguest<XXXX>@midway2.rcc.uchicago.edu`
  Push the button when asked for password
- Please, do not forget to return it at the end of the class

Login to midway: ssh

- A standard way to login to a remote Linux machine is to use ssh - secure shell. For example, to log into RCC's midway 2 cluster:

  ```
  ssh -Y <youruserid>@midway2.rcc.uchicago.edu
  ```

- The `-Y` option allows to use not only command line interface (CLI) but also graphics (`X-windows`) remotely - X-forwarding
- This assumes that you have ssh client on your laptop.
- If you run Linux on your laptop, you have everything but then you probably would not be here
- If you use Mac, than you should have ssh client although for X-forwarding you might need to install X11 server and client libraries, which can be downloaded from the XQuartz project page: `https://www.xquartz.org`

Login to midway: ssh

- If you have MS Windows, you might need to install a client. For example:
  - Putty:
    - `http://www.putty.org`
    - considered standard ssh client under MS Windows
    - does not allow X-forwarding
  - Bitvise:
    - `http://www.putty.org`
    - more advanced than Putty, available from the same page
    - does not allow X-forwarding
  - Cygwin:
    - `http://www.cygwin.org`
    - allows you to run most UNIX commands inside MS Windows - might be too much if you just want ssh client
    - contrary to the above ssh clients for Windows, it allows X-forwading if you install X11 server
    - It might take several hours and gigabytes to install the full Cygwin distribution but it can include everything: C/C++ compilers, python, perl, MySQL and Postgres databases, etc.

Login to midway: ssh

- MobaXterm:
  - `https://mobaxterm.mobatek.net`
  - allows X-forwarding
- If you have Chromebook, or anything else running Chrome web browser, you can use ssh extension to Chrome
  - does not allow X-forwarding
  - should work on any OS if it has Chrome

Login to midway: ThinLinc

- ThinLinc is portable on the client side in the sense that it should work with any OS on your laptop as long as you have a web browser. For many OSes you can also install client.
- It does not work with yubikeys
- ThinLinc server is a commercial product that might be running on some Linux sites but is not a standard part of Linux, contrary to ssh server, and is not used in most other places. At RCC we run it on midway 1 & 2 clusters but not on Hadoop cluster.
- So eventually you would need to get used to ssh anyway.

Login to midway: ThinLinc

- There are two ways to connect with ThinLinc to midway:
  - Just use a web browser interface by pointing your browser to `https://midway2.rcc.uchicago.edu`. This might not work well for all browsers, you might have to try several: Chrome, Firefox, Safari...
  - Install ThinLinc client from `https://www.cendio.com/thinlinc/download`.
    - Configure the client to connect to `midway2.rcc.uchicago.edu`
    - You can specify the dimensions of the window in which it is running. I am usually using full screen.
    - It does X-forwarding and you can use graphics, usually it works faster than vanilla ssh.
    - The client has trouble with emacs key sequences, web interface does not.

Login to midway: JupyterHub

- Yet another non-standard but free way to log into midway cluster is to use JupyterHub.
- There are two JupyterHub servers that you can use:
    - `https://jupyter.rcc.uchicago.edu` - midway 1
    - `https://hadoop.rcc.uchicago.edu` - Hadoop
- Just point your browser and use your midway username and password
- JupyterHub, besides running notebooks in python, R, and other interpreters, gives you a terminal and an ability to upload/download files
- It does not allow X-forwarding
- Since all RCC clusters share home file system, everything uploaded via any of the above JupyterHub instances becomes accessible from all the clusters
- There is no JupyterHub on midway 2 but once you get to a terminal, you can ssh there if necessary.

Login to midway: 2fa

- Since the end of last year, there is an extra extremely annoying security complication: 2-factor authentication.
- You need to follow the instructions on `https://cnet.uchicago.edu/2FA/index.htm` to enroll in 2-factor authentication. Another useful related page: `https://2fa.rcc.uchicago.edu`.
- You would need to install an application on your phone so that when you try to connect to midway, you get notification on your phone to which you need to respond
- If you do not have a phone or travelling abroad, you can get a set of passcodes that you can type instead, each time using the next passcode from the list and periodically getting a new list once you use all the numbers from there.

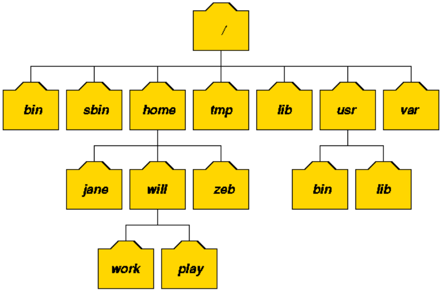
Lab 1: connect to midway 2

- Connect to midway 2 using one or more of the described methods, if possible with X-forwarding
    - 15 minutes or as much time as necessary for everbody to get onto midway 2 since the rest of the tutorial relies on your ability to execute commands there.
- Once you are on midway, get the presentation and labs from git repo:

```
git clone https://git.rcc.uchicago.edu/ivy2/Linux.git
```

Navigating the file system: directory tree

- Linux, like any other OS I have seen so far, organizes directories and



  files into a tree
- At the top of the tree there is root `/`
- The linux system commands, like ls, cp, mv, are typically in `/bin`
- Application programs, like emacs, vi, find, are typically in `/usr/bin`
- Commands for system administrators are usually in `/sbin`
- Libraries are usually in `/lib`, `/lib64`, `/usr/lib`
- User home directories are under `/home`
- Home directory of the root (system administrator user) is in `/root`

Navigating the file system: directory tree

- Many programs write temporary files into `/tmp`
- Various services often keep log files in `/var/log`
- Configuration files for the system and various services are typically under `/etc`
- To refer to a particular directory you can either use `absolute path` or `relative path`:
  - When using an absolute path, you construct it from the top of the directory tree. For example: `/home/ivy2/dir1/dir2/dir3`
  - When using a relative path, you construct it from your current location.
    - For example, if you are in `/home/ivy2/dir1` you can refer to the above directory as `dir2/dir3`.
    - If you are already in `/home/ivy2/dir1/dir2/dir3`, you can refer to the current directory as `.`. Sometimes it is necessary: for example, when you want to execute a program from the current directory and do not have `.` in `$PATH`: `./myprogram`.
    - If you are in `/home/ivy2/dir4/dir5/dir6`, you refer to the above directory as `../../../dir1/dir2/dir3`. `..` denote a parent directory.

Navigating the file system: directory tree

- To list what files and subdirectories are in a particular directory and learn their properties, such as size, permissions, ownership, last date of modification, use `ls`
- To change directory, use `cd`
- To see what directory you are in, use `pwd`
- To make directory, use `mkdir`
- To remove a file or directory, use `rm`
- To create an empty file, for example, for testing purposes, `touch`
- To find a file or directory under some subtree, use `find`
- To find an exact syntax, options, try `man` or `--help` option.

Navigating the file system: Lab 2

- `cd $HOME/Linux/labs/2`
- Follow instructions in `README.md`

File editing: emacs

- `emacs [<filename>]` starts GUI version if X is forwarded, otherwise, it either starts terminal version or complains
- `emacs -nw [<filename>]` starts terminal version
- Just type your text
- To cut everything from the current character to the end of line: `Ctrl+k`
- To paste: `Ctrl+y`
- To save a file: `Ctrl+x Ctrl+s`
- To save and exit: `Ctrl+x Ctrl+c`
- To mark a big part of text, `Esc Shift+@`, move cursor, do something with the selected text, for example `Ctrl+w` to cut it.
- To search text forward for some string `Ctrl+s`, to search backward `Ctrl+r`
- To move cursor to the end of the line `Ctrl+e`, to move cursor to the beginning of the line `Ctrl+a`

File editing: emacs

- To replace a string starting from the current character till the end of file: `Esc+x replace-string`, type source and target.
- You can open several files inside emacs and keep switching between different buffers, copying and pasting between different files.
- To open an existing file or start a new one in emacs `Ctrl+x Ctrl+f`
- To list open files (buffers): `Ctrl+x Ctrl+b`
- To switch between different buffers: `Ctrl+x b`
- One can open multiple windows, each looking at the same or different buffers.
- To open a new window below: `Ctrl+x 2`
- To open a new window to the right: `Ctrl+x 3`
- To remove other windows: `Ctrl+x 1`
- To move to other window: `Ctrl+x o`
- One can cancel a command in construction with `Ctrl+g`

File editing: emacs

- By default emacs shows line number on the bottom bar. One can ask it to show also column number with `Esc+x column-number-mode`
- To undo, `Ctrl+x u`
- There are thousands of other commands and options but the above is typically enough for 99% of text editing
- You can define your own macros
- Emacs usually detects what language you are programming in and formats text accordingly. It can recognize C, C++, Fortran, Python, Java, HTML, LaTeX, etc. Language specific menu appears in GUI version of emacs.
- The other popular editor in Linux which we do not discuss here is `vi`.
- There are many other higher level and more user friendly editors like gedit, nano, lyx, libreoffice, atom, etc. but you better know either emacs or vi since those are available everywhere.

File editing: gedit

- If you do not want to spend a few hours to get used to the power of emacs, just use gedit
- It has similar intuitive GUI interface to Window's notebook, nothing to learn
- However, you need either to use ssh client that does X-forwarding or use ThinLinc

File editing: Lab 3

- `cd ~/Linux/labs/3`
- Follow instructions in `README.md` file

Copying and moving files and folders

- To copy files on a Linux system, use `cp`, to copy folders, use either `cp -r` or `rsync -av`
  - `cp <fromfile> <tofile>`
  - `cp -r <fromdirectory> <todirectory>`
  - `mv <fromfile> <tofile>`
  - `rsync -av <fromdirectory> <todirectory>`
- `rsync`, contrary to `cp`, is restartable, if it is interrupted in the middle and you start it again, it figures out what has already been done and picks up where it left.
- To copy file/directory to remote host, if you have ssh client, do

```
scp <fromfile> <yourusername>@midway2.rcc.uchicago.edu:<tofile>
scp -r <fromdirectory> <yourusername>@midway2.rcc.uchicago.edu:<todirectory>
rsync -av <fromdirectory> <yourusername>@midway2.rcc.uchicago.edu:<todirectory>
```

- To copy file/directory from remote host, if you have ssh client, do

```
scp <yourusername>@midway2.rcc.uchicago.edu:<fromfile> <tofile>
scp -r <yourusername>@midway2.rcc.uchicago.edu:<fromdirectory> <todirectory>
rsync -av  <yourusername>@midway2.rcc.uchicago.edu:<fromdirectory> <todirectory>
```

- `rsync` uses `ssh` underneath unless `rsync` server is running on the remote site

Copying and moving files and folders

- If you have a lot of data to copy between your laptop and midway or between midway and some other Globus end point, use Globus. For instructions, see

  `https://globus.rcc.uchicago.edu/globus-app`
- If ssh/scp still does not work for you, you can use JupyterHub to upload/download data to/from midway

  `https://jupyter.rcc.uchicago.edu`
- If you are transferring text files from Windows/Mac to Linux or the other way arround, you might need to run `dos2unix` or `unix2dos` on those files since the end of line character in Linux and Windows/Mac text files is different which might cause problems for some programs.

Copying and moving files and folders

- Compare, using `cat -A <filename>` , the same text file edited on Windows and Linux
    - `windows.txt`:
      ```
      one two^M$
      three^Ifour^M$
      ```
    - `linux.txt`:
      ```
      one two$
      three^Ifour$
      ```
- `cat -A <filename>` displays non-printable characters denoting them in some way. For example, `^I` means tab, `$` - Linux end of line character.
- Look at the end of the lines. In Windows there is an extra `^M` before `$`.
- To convert Windows end of line character to Linux, run `dos2unix windows.txt`

Lab 4

- `cd $HOME/Linux/labs/4`
- Follow instructions in `README.md`

File and directory permissions

- For each file or directory you can set who can
    - r - read
    - w - write
    - x - execute (or browse for directory)

  it.
- For permissions purposes, there are four types of “who”:
    - u - user
    - g - group
    - o - others
    - a - all
- There is ACL (access control list) that gives finer control over permissions but I had never any reason to use it so far.
- There are permissions that each file and directory gets by default. The default is configurable.

File and directory permissions

- You can see permissions with ls -l:

  ```
  ls -l t8
  -rwxrw-r-x 2 ivy2 kicp 243640 Dec 11 09:38 t8
  ```

  The above says that a file `t8` was last modified on `Dec 11 09:38`, has `243640` bytes, belongs to `ivy2` user and `kicp` group (often default group is same as username), has only two hardware links to it, is a file (for directory the first character would be `d`, for symbolic links it is `l`), has `rwx` permissions for user, `rw-` permissions for the group, `r-x` permissions for others.
- The executable permission on a file means that it can be used as a program.
- The executable permission on a directory means that one can go there.

File and directory permissions

- To change permissions:

  ```
  chmod u-w t8
  chmod -R a+X dir1
  ```

- The above would remove write permissions for the user and makes all directories under dir1 browsable by everybody. If one used 'x' instead of 'X', all the files under dir1 in addition would become executable.
- The ownership of the file/directory is changed with chown but as a user you have rather limited freedom to do it. Only root can change user ownership. You can change group ownership to one of the groups of which you are a member:

  ```
  chown ivy2:rcc-hadoop t8
  ```

- You can find to which groups you belong by executing groups (standard Linux command) or `rcchelp user <username>` (local RCC script)

Bash shell

- When you log into Linux, you get into shell - command line interface (CLI) to Linux - where you can type commands and the shell would interpret and execute them.
- By default it is `bash`.
- There are other shells that you can select: `csh`, `tcsh`, `zsh`, etc. They mostly differ in syntax but have similar functionality.
- bash configuration is defined in the following files in your home directory: `.bash_profile`, `.bashrc`, `.bash_aliases`
- There you can set environmental variables, aliases, define, for example, how your shell prompt looks like, etc.
- If you start typing a command or a path in shell and press TAB, the shell tries to autocomplete
- `history` shows the previous commands, arrow-up returns the previous command if you want to repeat it
- bash understands many of emacs key combinations: `Ctrl+A`, `Ctrl+E`
- Shell can be used as a programming language to write scripts

Environment

- When you type the name of the executable program in a shell, it is looking for it among the paths listed in `$PATH` environmental variable. For example:

```
$ echo $PATH
/bin:/software/slurm-current-el7-x86_64/bin:\
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:\
```

- If you have a program in some non-standard location `/yet/another/path/bin/myprogram`, you need to prepend/append another path to `PATH` or use the full path to the executable:

```
$ /yet/another/path/bin/myprogram
$ export PATH=/yet/another/path/bin:$PATH
$ myprogram
```

- To find out where your program is, if its location is in `$PATH`:

```
$ which gcc
/usr/bin/gcc
```

Environment

- Notice that bash would go over the list of paths in order specified and will pick the first available executable with the given name it can find and stop looking after that. Therefore, it might make difference if you prepend or append paths.
- Most of the programs in Linux are dynamically compiled, in the sense that they load the necessary libraries at runtime rather then including them into the executable. If the library you are using is not at some standard location like `/lib64`, `/usr/lib`, `/usr/lib64`, `/usr/local/lib`, etc, you need to prepend/append the corresponding path to `$LD_LIBRARY_PATH` environmental variable.
- To see what libraries are loaded or not found for a particular executable:

```
$ ldd `which gcc`
linux-vdso.so.1 =>  (0x00007ffdbcba1000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff4b95a6000)
/lib64/ld-linux-x86-64.so.2 (0x00007ff4b9970000)
```

Environment

- The program behavior might be modified by various environmental variables specific to the program.
- For example, programs that use OpenMP multithreading would take the number of threads to use from `$OMP_NUM_THREADS`
- You can learn the path to your home directory, username, current directory:

  ```
  $ echo $HOME
  $ echo $USER
  $ echo $PWD
  ```
- This is useful for writing portable programs that do not hardcode such information but learn it on the fly from the environment.
- To see the values of all the current evironmental variables use `printenv`
- If you use python and have some python libraries installed separately from python distribution, add the corresponding directory to `$PYTHONPATH`

Modules

- If you are running Linux on a desktop or laptop, you would typically have one version of each program installed in some standard directory
- In HPC environment, like midway, this is unrealistic: different users want different versions of the same program.
- As a result, applications are typically installed in a non-standard location and one must set environmental variables, at least `PATH` and `LD_LIBRARY_PATH` to select a particular version of a particular program
- To simplify this task we use `modules`
- You can see which modules are available by running `module av`
- You can load the module with `module load <name>/<version>`.
- You can unload the module with `module unload <name>/<version>`.
- If you do not specify version, default is used
- To unload everything, use `module purge`.

Modules

- For example:

```
which gcc
module load gcc/7.2.0
which gcc
module unload gcc/7.2.0
which gcc
```

- To see what a module does, use `module show <name>/<version>`. For example:

```
[ivy2@midway2-login1 ~]$ module show gcc/7.2.0
-------------------------------------------------------------------
/software/modulefiles2/gcc/7.2.0:

module-whatis   setup gcc 7.2.0 compiled with the system compiler
conflict        gcc
prepend-path    PATH    /software/gcc-7.2.0-el7-x86_64/bin
prepend-path    LD_LIBRARY_PATH /software/gcc-7.2.0-el7-x86_64/lib
prepend-path    LIBRARY_PATH    /software/gcc-7.2.0-el7-x86_64/lib
prepend-path    LD_LIBRARY_PATH /software/gcc-7.2.0-el7-x86_64/lib64
prepend-path    LIBRARY_PATH    /software/gcc-7.2.0-el7-x86_64/lib64
prepend-path    CPATH   /software/gcc-7.2.0-el7-x86_64/include
prepend-path    MANPATH /software/gcc-7.2.0-el7-x86_64/share/man
-------------------------------------------------------------------
```

Lab 5

- `cd $HOME/Linux/labs/5`
- Follow instructions in `README.md`

Input/Output redirection and pipes

- You can redirect output from one program, standard output, to a file:
```
ls -l > /tmp/my.out; pwd >> /tmp/my.out; cat /tmp/my.out
```
- Or you can use the output from one program as an input to another program with pipes. Most of Linux commands are written as filters that can take input either from a file or standard input, transform it, and print the results to standard output. For example:
```
ls -l | sort
cat /tmp/ls.out | grep test | grep -v something | tail -2
grep test < /tmp/ls.out > /tmp/ls_1.out
```
- If a program prints out to a screen an error or a warning, it is often done to a different stream, standard error, that can be treated separately from standard output and can be redirected to a separate file or merged with standard output. If you do not redirect it somewhere, it will be printed to the screen.
```
nohup ./myprogram > myoutput 2>myerror &
nohup ./myprogram > myoutput 2>&1 &
```

Lab 6

- `cd $HOME/Linux/labs/6`
- Follow instructions in `README.md`

Process control

- ps - reports a snapshot of the current processes.
- There are hundreds of options but usually I use only two:

  ```
  ps
  ps -ef | grep <...>
  ```

- top - provides a dynamic real-time view of a running system, sorts the processes by load, memory usage, etc
- If you need to kill a running process, find its pid with either ps or top and then

  ```
  kill <pid>
  ```

  or, if it does not work, as the last resort

  ```
  kill -9 <pid>
  ```

Process control

- `&` - if you put it at the end of the command, it is executed in the background and you can continue using the shell; however, would not persist after you disconnect from the terminal

  ```
  <program> &
  ps
  ```

- `nohup` - execute a program in a background, it would still run after you disconnect

  ```
  nohup <yourprogram> > <output> 2>&1 &
  ```

- You can put the existing foreground program into background with `Ctrl+Z`
- To bring it back into foreground, `fg`
- It is sometimes useful to insert into your scripts

  ```
  sleep <N>[s|m|h]
  ```

  to make the program sleep for N seconds/minutes/hours

Miscellaneous useful commands

- `grep` - search for strings in a file
- `sort` - sort text
- `uniq` - eliminate duplicate lines in text
- `head`, `tail` - show first/last *n* lines in a text file
- `history` - list previous commands
- `cut` - cut columns from a tabular file
- `paste` - merge columns from different files
- `alias` - define an alias for a command or combination of commands
- `cat` - print a file to screen
- `more`, `less` - show a text file, page by page
- `gzip` - compress file
- `tar` - archive directory tree into a file

Wildcards

- `ls z*.out` - show a list of files in the current directory that start with `z` and end with `.out`. There can be any number of characters in between or nothing
- `ls t?at` - show a list of files that start with `t`, end with `at` and have one character in between

Shell programming

- Shell can be used as a programming language
- You can set environmental variables, define aliases, functions, list commands to execute
- There are flow control structures like if conditional statements, loops, functions, etc.
- I usually just use shell to set up environment, aliases, to list commands; for more complicated scripting I would rather use perl or python which are much more convenient to program
- To get command line arguments into shell script:
  - `$0` - name of the file with script
  - `$1` - name of the first argument, etc.
- If you want to know more about bash shell programming, take a look at:

  `http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html`
  `http://tldp.org/LDP/abs/html/`

RCC resources

- Research Computing Center has the following clusters:
  - midway 1:
    - $\sim$ 1100 compute nodes, 2 login nodes,
    - Sandy Bridge CPU, 16 cores in each node,
    - a typical node has 32G of RAM, there are a few big memory nodes with up to 1T of RAM;
    - Scientific Linux 6.9
    - FDR infiniband network
  - midway 2:
    - $\sim$ 400 compute nodes, 2 login nodes,
    - Broadwell CPU, 28 cores in each,
    - a typical node has 64G of RAM, there are a few big memory nodes with up to 512G of RAM;
    - 6 GPU nodes with 4 Tesla K80 cards in each;
    - Scientific Linux 7.2
    - FDR & EDR infiniband network
  - Hadoop cluster: 3 management nodes, 4 compute nodes, 200T of HDFS file system, 10G network.

RCC resources

- All clusters share GPFS file systems in `/home/<username>`, `/project`, `/project2`, `/scratch/midway2/<username>`.
- midway 1 has two login nodes: `midway-login1.rcc.uchicago.edu` and `midway-login2.rcc.uchicago.edu`
- You can either explicitly `ssh` to one of those or `ssh` to `midway1.rcc.uchicago.edu` and let the load balancer put you on to one of the login nodes.
- midway 2 has two login nodes: `midway2-login1.rcc.uchicago.edu` and `midway2-login2.rcc.uchicago.edu`

RCC resources

- You can either explicitly ssh to one of those or ssh to `midway2.rcc.uchicago.edu` and let the load balancer put you on to one of the login nodes.
- Login nodes are to be used mostly to submit your program for execution on the compute nodes
- You can also edit and compile your program on login nodes, run small tests and postproduction tasks.
- If login nodes get overloaded (either by reaching load or memory threshold, $\sim 90\%$), the user responsible for the largest fraction of the load is found and all his processes on the login node are killed.
- Heavy calculations should be done on compute nodes.

RCC resources

- midway clusters use Slurm resource manager to schedule jobs on compute nodes
- More details about RCC software and hardware can be found in `http://rcc.uchicago.edu`
- Some other useful midway-related links:

```
If you want to run Jupyter on compute nodes:
https://git.rcc.uchicago.edu/ivy2/Jupyter_on_compute_nodes
https://git.rcc.uchicago.edu/ivy2/Troubleshooting_JupyterHub
My Hadoop and Spark tutorials:
https://git.rcc.uchicago.edu/ivy2/Graham_Introduction_to_Hadoop
https://git.rcc.uchicago.edu/ivy2/Spark
Instructions how to run various Deep Learning frameworks on midway:
https://git.rcc.uchicago.edu/ivy2/TF_midway2
```

- Let us learn how to submit jobs to Slurm on midway 2.

Submitting jobs to Slurm: batch

- The standard way of running big jobs on big HPC system is to submit your job to the scheduler. The scheduler would put your job into queue and run whenever possible depending on the load on the cluster, the amount of resources you asked, your priority, etc.
- One prepares a submit file that:
  - Requests resources such as memory per core, number of nodes, number of cores in each node, number of GPU cards
  - The submit file also specifies which allocation on the cluster to use, in what partition to run, name of the log files
  - All the above is done at the beginning of the file on lines that start with `#SBATCH`
  - Next one sets an environment by loading some modules
  - Finally one specifies one or more executables to run
  - Slurm defines many useful environmental variables to help you script the submit files
  - For details, see
    https://slurm.schedmd.com/
    https://rcc.uchicago.edu/docs/running-jobs/index.html

Submitting jobs to Slurm: batch

- The submit file is sent to the scheduler with sbatch command which takes options that can overwrite some of the options specified in the submit file

  ```
  sbatch my.batch
  ```

- Job id number is printed. You can use it to monitor job progress with squeue

  ```
  squeue -j <jobid>
  ```

- Or you can list all your jobs:

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

- One can kill the job with

  ```
  scancel <jobid>
  ```

- When your job is running, you can ssh to the corresponding node if you need to diagnose what is going on.
- Graham school has a dedicated GPU partition `-p mscagpu -A mscagpu` and dedicated disk space `/project/msca/<username>`

Lab 7

- `cd $HOME/Linux/labs/7`
- Follow instructions in `README.md`

Submitting jobs to Slurm: interactive

- If you want to work interactively on a compute node, you can do this with sinteractive that takes the same command line options as batch
- `sinteractive -p gpu2 --gres=gpu:1 --time=5:00:00`
- sinteractive does X-forwarding and you can use graphics on the compute node
- Both with sbatch and sinteractive jobs you might have to use `-A` option to specify the allocation you are charging your job to.
- Also, there might be special reservations made during your class to insure that you get a node. In that case you would need to use `--reservation=<reservation name>` option.
- If you want to ask for all the cores and memory in the node, instead of `--ntasks-per-node=28`, you can just use `--exclusive`
- If you have such a job that runs simultaneously on different nodes and different processes need to communicate, consider using `MPI`
- If your job can utilize multiple threads on CPU cores, use `OpenMP`
- For GPU programming, consider `OpenACC` and `CUDA`

Appendix: Tutorials on demand

- Currently the following tutorials are available on demand:
  1. Introduction to Linux and RCC
  2. Introduction to Python
  3. Distributed MPI programming in C/C++/Python
  4. Multithreaded programming in OpenMP
  5. GPU Programming with CUDA
  6. Introduction to Hadoop
  7. Big Data and Machine Learning with Spark and BigDL
  8. Deep Learning with Keras and TensorFlow
  9. Deep Learning with Theano
  10. Deep Learning with PyTorch
  11. Sequence Models with Recurrent Neural Networks
  12. Reinforcement Learning
  13. Autoencoders
  14. Introduction to Elastic Search
  15. Singularity Container
- More are coming: GPU Programming with OpenACC, C++, Generative Adversarial Networks, Object detection with YOLO