Linux Tutorial

Introduction

Unix is an “old-school” operating system, developed at Bell Labs in the 1970s — before the mouse, menus, windows, Macs and PCs. Linux is an open-source version of Unix. Under Unix, you communicate with the computer by *typing commands*. This sounds primitive, but it has survived because it lends itself to automation. Typed commands can be put in a text file and executed as a “script”.

Unix commands have short names — cd, pwd, ls — to save typing. These names are usually abbreviations: cd = Change Directory, pwd = Print Working Directory, ls = LiSt files. To remember the command, learn the unabbreviated name.

Unix operates under a philosophy of “no news is good news” and “no handholding the user”. You type a command and Unix does it, often silently. You can’t “see” that it did what you asked. If you type “rm *” (DON’T! — this deletes all files in the current folder) Unix complies, without asking “are you sure?”.

bash The “shell” is the program you’re using when you’re not using a program. The default shell on our research computers is “bash”. Bash has many commands and features. You can write scripts in bash — short programs that automate repetitive tasks.

man (Manual.) Help in Unix is primitive. It is vital to know the names of common commands. If you know the name, you can get help on how to use it by entering “man <command>”. (In these notes, <…> means “replace this with something relevant”.) Within man, spacebar or “ctrl-F” goes forward a page, “ctrl-B” backwards, and “q” quits.

Man pages are hard to read. Internet searches like “how to <something> in Unix” are often helpful.

options Commands have many options, which are key to using them. Some options are commonly used, most are not relevant to you. An example of a command (ls) with an option (l) is “ls -l”, which means (see below) list files in current folder (ls) in a “long form” (-l). Options can be “stacked”, as in “ls -l -a” or “ls -la”.

ctrl-C Sometimes Unix responds to a command by pouring out text or taking forever or doing something you didn’t intend. You can terminate the command with “control-C”.

history Unix takes a lot of typing. Sometimes you mistype a command, or want to change it. You can make previously entered commands reappear with the up-arrow key. The previous command can be edited (by moving the cursor back, deleting and/or retyping) and then re-entered.

Login and logout

To log into the cluster, you need a terminal program running on your desktop or laptop. On Windows, use puTTY; on Mac, Terminal or iTerm (an improved version of Terminal). Terminal is installed by default on Mac; both puTTY and iTerm must be installed. puTTY is available at https://www.putty.org, iTerm at https://iterm2.com.

ssh (Secure SHell.) "ssh -X <user>@<machine>" logs <user> onto <machine>. Option -X is for "X forwarding", which lets Unix show graphics on your laptop. The machine name for Collab is submit.hpc.psu.edu.

You will be prompted for your password, and then for two-factor authentication (2FA), which asks for confirmation on your smartphone that you are you. To set up 2FA, see https://accounts.psu.edu/2fa

To use any application that "opens a window" (called an "X11" application), such as Mathematica, you need an additional application. On the Mac, this is XQuartz, available at https://www.xquartz.org. On the PC, you need VcXsrv, available at https://sourceforge.net/projects/vcxsrv/.

An alternative way to log onto the cluster, which provides more of a "PC feel" with desktops and menus, is to launch an "interactive batch session". This is accessed from a webpage https://rcportal.hpc.psu.edu/pun/sys/dashboard, with the icon "Roar Collab RHEL8 Interactive Desktop".

exit "exit" ends a terminal session.

Text editors

Unix is a text-based operating system; programs, batch scripts, and parameter files are text files. To work effectively on Unix, you need a good text editor, that you become facile with. There are several options.

gedit gedit is a modern windows-style text editor for Unix, with cursor / mouse / cut-and-paste / menus. It is intuitive for Windows and Mac users, and does not require key commands to navigate the file.

vi (VIsual editor.) For old-school users (or those who appreciate old-school wisdom), the editor "vi" works well. It is "visual" in that you can see the contents of the file. However, you don't navigate in the file with cursor and mouse, but by key commands. It offers powerful search-and-replace functions, and rapid navigation within the file. Learning to use vi is analogous to learning to touch-type — difficult at first, but once learned is remarkably fast.

For vi tutorials, see:

https://www.cs.colostate.edu/helpdocs/vi.html
http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Editors/ViIntro.html
http://www.viemu.com/a_vi_vim_graphical_cheat_sheet_tutorial.html.
Once you know vi, this summary is useful: https://bpb-us-e1.wpmucdn.com/sites.psu.edu/dist/0/79295/files/2020/09/notes-on-vi.pdf.

Files can be edited on the laptop and transferred to the cluster; evidently this is awkward for frequent small changes. If you do this, you must use a *text editor*, not a word processor (like Word), which introduces hidden special characters that make a mess of a text file. On the Mac, a good choice is BBEdit https://www.barebones.com/products/bbedit/; on the PC, the best alternative is Notepad++.

Navigating the file structure

Like Windows or OS X (Macintosh operating system), Unix files are arranged in a "file structure" of folders, which can contain other folders. With no visual representation of files and folders, you navigate the file structure with typed commands.

The "name" of a folder (called a "directory" in Unix) looks like "/storage/home/stm9". Directory "stm9" is in "home", which is in "storage", which is in the "root directory" (indicated by the first "/"). A directory name is also called a "path" (it's a path to a group of files).

pwd (Print Working Directory.) Prints the directory you're currently in.

cd (Change Directory.) "cd <directory>" takes you to <directory>. If you're in "/storage/home" which contains "stm9", "cd stm9" works. "cd" by itself takes you to your $HOME directory (see below).

.. and . ".." means the "parent directory" — one level above where you currently are. "." means the current directory itself (for why you need that, see below).

ls (LiSt files.) "ls" produces a list of files in the current directory. "ls -l" gives more information — the file permissions (see below), owner, size (in MB), and creation / modification date.

hidden files Files with names beginning with "." are "hidden" — not shown by ls or ls -l. They contain settings for various programs and are not meant to be edited, except for .bashrc (see below). "ls -a" lists all files including hidden.

Files and directories

cp (CoPy file.) "cp <file> <target>" makes a copy of the file, with name "target". If "target" includes a path, like "cp thisfile ../newfile", then newfile will reside in ".." (here, the parent directory).

mv (MoVe file.) "mv <file> <target>" moves the file (no copy), to the location "target". This can be used to rename a file: "mv oldName newName" changes the name. If "target" is a path, the file keeps its name but is moved.

rm (ReMove file.) "rm <file>" deletes the file. "rm -r <fileDescriptor>" acts *recursively* on all subdirectories of the current directory, deleting everything that matches <fileDescriptor>. "rm -r *" deletes *everything* in the current folder. Use with caution.

mkdir (MaKe DIRectory.) "mkdir <newDirectory>" makes a new directory with the specified path.

rmdir (ReMove DIRectory.) "rmdir <directory>" removes the directory — which must be empty of files to work.

Wildcards and patterns

For any command that takes one or more files or directories as an argument, we often want to apply the command all at once to one or more files with similar names. Unix includes a way to specify a *pattern* (called a "regular expression") that can "match" one or more filenames. These patterns are used by many Unix commands that manipulate files and character strings.

A simple example is "*" (asterisk), which by itself matches anything (any string of zero or more characters). Hence "mv * .." moves *all* files to the parent directory; "rm demo*.nb" deletes all files that start with "demo" and end in ".nb" (i.e., have "file extension" nb).

Unix patterns (also called "regular expressions") have many options beyond the simple use of "*"; see https://ryanstutorials.net/linuxtutorial/wildcards.php.

File attributes

permissions Files and directories in Unix have "permissions", which limit who can read (r), write (w), or execute (x) the file or directory contents. There are permissions for the file *owner*, for the *group* of users assigned to the file, and for *others*. Permissions can be seen with "ls -l <file>".

chmod (CHange MODe.) The "mode" of a file means its permissions. For example, "chmod +x" makes a file executable by anyone; "chmod -w" removes write access by anyone. The general form is
"chmod <+ or -><who><what>", where "who" is "u" (you, the owner), "g" (group), or "o" (others), and "what" is "r" (read), "w" (write), or "x" (execute).

touch "touch <file>" resets the created/modified date of a file to the current date, without altering the file. This is useful to prevent files on /scratch from being automatically deleted (with no backup!) when they are more than 30 days old.

Looking for files

Operating systems like OS X have advanced, easy-to-use tools for finding files on your crowded hard drive. Unix has old-school command line tools.

which "which <file>" looks for <file> on all the directories in your PATH (see "PATH" below). It is used to search for executable files, to see if they are installed and you have access to them.

The following tools have many options, and require examples to understand how to use them:

grep (Global Regular Expression Parser.) "grep <pattern> <file>" is used to search a text-containing file for a particular text string, or "pattern" that matches a string of text. See https://alvinalexander.com/unix/edu/examples/grep.shtml

find find looks for files by searching over the directory structure. It is complicated to use but powerful. See https://alvinalexander.com/unix/edu/examples/find.shtml

Environment

Unix uses "environment variables" to store information used by various programs (including the shell). "$<variable>" is the value of <variable>. Two key variables are:

HOME The path to your "home directory", where you are when you first log in. Don't change HOME.

PATH The list of paths where the shell looks for executable files ("programs"). To add "/usr/local/bin" to PATH, type "PATH=$PATH:/usr/local/bin".

echo "echo $HOME" prints the value of HOME. More generally, echo prints out the value of its arguments. (It is often used in scripts, see below.)

alias A way to save typing is to "define new commands" with alias.
"alias collab=`ssh -X stm9@submit.hpc.psu.edu'" makes "collab" an "alias" for the command after the equals sign. "alias" by itself prints all current aliases. Aliases you define manually disappear when you log out.

module Not all cluster software is available to you when you log in.
"module load <program>" — loads <program>
"module avail" — lists modules available to load
"module list" — lists currently loaded modules
"module spider <program>" — searches for <program> to load, and prints *instructions* if other modules must be loaded first. (Long output advances with spacebar, like man.)

.bashrc When you log in, commands in the file "$HOME/.bashrc" are automatically executed. Lines may be added to .bashrc to do useful things, like defining frequently used aliases, or loading frequently used modules.

Redirection and pipes

standard input/output Unix commands and simple programs take input from "standard input" and direct output to "standard output". If you do nothing, standard input is what you type in, and standard output is what the program prints out.

Standard output can be sent to a file with *redirection*. This avoids a "Save File As…" dialog box and makes automation easier. Redirection can also get standard *input* from a file instead of the keyboard.

Standard output can also be rerouted to the input of another program with a *pipe.* This allows programs to work together in a variety of useful ways.

redirection To redirect output of a program to a file: "ls -l > fileList" saves the output of "ls -l" in the file "fileList". To append to an existing file, use ">>". To get standard input to a program "foo" from a file "fooInput", use "foo < fooInput".

pipes To direct the standard output of one program into the standard input of another program, use a "pipe". A simple example is "cat <veryLongFile> l more". "cat" (conCATenate) copies a text file to standard output. "more" is a simple program that prints its standard input with controls like "man" (spacebar for next screen, q to quit, etc.).

Working in background

Sometimes in an interactive session you want to execute a Unix command or program that will take a while to complete. To avoid waiting, you could open another terminal window and log in again. Or, you can execute the command in the *background.*

To do this, type "<slowcommand> &". The final & character forces <slowCommand> to run in the background. You get a new command prompt and can continue working.

If the command output is not redirected, it will appear intermittently on the screen as you work.

Without special steps, background process do *not* keep running after you logout. (If that's what you want to happen, you should be submitting a batch job, see below.)

ps ps (ProcesS) lists the running processes including background processes, and reports their PID (Process ID), run time so far, and command name.

kill "kill <PID>" terminates a background process.

File transfer

File transfer to other computers is done using sftp (Secure File Transfer Protocol).

nccise (the Milner group cluster) and Collab have separate filesystems, so file transfer between them requires sftp.

sftp (Secure File Transfer Protocol.) "sftp" launches an interactive program with its own command prompt. "open <machine>" connects to <machine>; you will be prompted for your password and 2FA. From Collab, nccise is "nccise.dc.psu.edu"; from nccise, Collab is "submit.hpc.psu.edu".

Within sftp, commands like pwd, cd, cp work much as usual. To navigate locally, use the same commands with "l" prepended: lpwd, lcd, lls. For help see https://www.tecmint.com/sftp-command-examples/

to/from laptop To transfer files between the cluster and your own machine, there are two options: 1) you can use sftp applications on your laptop; for OS X, use Cyberduck https://cyberduck.io/?l=en. For PC, use WinSCP https://winscp.net/eng/index.php.
2) you can use the Collab portal, https://rcportal.hpc.psu.edu/pun/sys/dashboard, which offers a web interface to your files on Collab (top menu, Files).

tar (Tape ARchive.) tar is the Unix equivalent of the "Compress Items…" command in the OS X Finder menu. It takes many files and creates a single "archive" file (called a ".tar" file or "tarball"). For examples see https://www.tecmint.com/18-tar-command-examples-in-linux/

Globus. Globus is a web-based interface for file transfer between different filesystems, including Collab and OneDrive, as well as machines outside Penn State. For information, see https://docs.globus.org/how-to/get-started/. Globus transfers between "endpoints"; the endpoints for Collab and OneDrive are "PennState_ICDS_RC" and "Penn State OACIOR OneDrive Collection 01".

Batch queue

Big computing jobs that take a long time are what the cluster is for. You don't logon and wait for such jobs to finish; instead, use the "batch queue".

nccise and Collab use different queue managers: nccise uses pbs (Portable Batch System), Collab uses SLURM (Simple Linux Utility for Resource Management). The two utilities do basically the same thing, but the commands and options are completely different.

PBS

Three basic commands interact with the batch queue:

qsub (Queue SUBmit.) “qsub <script>” submits the script file <script> for batch execution. The script file (see below) contains directives to PBS (the Portable Batch System), with information like: which queue to submit to, maximum run time, number of CPU cores requested, memory requested, and the commands or programs to be executed.

qstat (Queue STATus.) “qstat -u <username>” reports on status of your submitted jobs. “qstat <queue>” reports on status of all jobs in <queue>. This is useful for seeing how busy a given queue is.

qdel (Queue DELete.) “qdel <jobID>” kills a queued or running job. The jobID values are reported by qstat -u.

For help with batch queue options (PBS commands), see
https://albertsk.files.wordpress.com/2011/12/pbs.pdf

SLURM

The corresponding three commands under SLURM are:

sbatch. “sbatch <script>” submits the script file <script> for batch execution.

squeue. “squeue -u <username>” reports on the status of your submitted jobs.

scancel. “scancel <jobID>” kills a queued or running job.

For more help with SLURM, see
https://hpc.nmsu.edu/discovery/slurm/commands/ and
https://docs.crc.ku.edu/how-to/slurm-options/.

Batch scripts

To use the batch queue effectively requires scripts. Scripts are short programs, written in a “scripting language”. The two script languages our group uses are bash (the bash shell can execute scripts of bash commands) and python. Bash is for simpler scripts; python is more powerful.

Batch scripts all do the same set of things: setting options for the batch queue, passing arguments to the program to be executed, and naming input and output files in an organized way. More complicated batch scripts typically vary some set of parameters in a systematic way over a series of submitted jobs. The best way to learn about them is to study working examples.

Gromacs batch script example

Here is a typical PBS batch script for Gromacs:

```
#!/bin/bash
#PBS -A stm9_m_gpu
#PBS -l nodes=1:ppn=8:gpus=1
#PBS -l walltime=4:00:00
#PBS -l mem=1gb
#PBS -N 8c1g

module load gromacs-2019.6

TEST=dme2000
cd $PBS_O_WORKDIR
MDP=$PBS_O_WORKDIR/MDP
SYSTEM=$PBS_O_WORKDIR/System
EQUIL=$PBS_O_WORKDIR/Equil8

mkdir 8c1g
cd 8c1g

gmx grompp -f $MDP/nvt.mdp -c $EQUIL/min.gro -p $SYSTEM/$TEST.top
  -o nvt.tpr -maxwarn 3
gmx mdrun -nt 8 -gpu_id 0 -deffnm nvt
```

Comments:

0) Omit “module load gromacs” if your .bashrc includes it
1) “ppn” = processors per node
2) “-A” is the account (not used on nccise)
3) “-N” names the job
4) $PBS_O_WORKDIR = where you submitted from

The same batch script translated to SLURM would read:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --gpus=1
#SBATCH --mem=1gb
#SBATCH --time=4:00:00
#SBATCH --job-name=8c1g
#SBATCH --account=stm9_m_gpu
#SBATCH --partition=sla-prio

module load gromacs-2019.6

TEST=dme2000
cd $SLURM_SUBMIT_DIR
MDP=$SLURM_SUBMIT_DIR/MDP
SYSTEM=$SLURM_SUBMIT_DIR/System
EQUIL=$SLURM_SUBMIT_DIR/Equil8

mkdir 8c1g
cd 8c1g

gmx grompp -f $MDP/nvt.mdp -c $EQUIL/min.gro -p $SYSTEM/$TEST.top
  -o nvt.tpr -maxwarn 3
gmx mdrun -nt 8 -gpu_id 0 -deffnm nvt
```

Comments:

0) Omit “module load gromacs” if your .bashrc includes it
1) “ntasks” = number of cores (“ppn” in PBS)
2) “partition=sla_prio” if you are using my account (“stm9_m_gpu”)
3) for open queue, “account=open”, and omit “partition”
4) $SLURM_SUBMIT_DIR = where you submitted from (“$PBS_O_WORKDIR” in PBS)