Lab 7:
Basics of Command Line and Unix
and
Familiarity with Phasers

Instructor: Vivek Sarkar

Course wiki : https://wiki.rice.edu/confluence/display/PARPROG/COMP322

Staff Email : comp322-staff@mailman.rice.edu

Goals for this lab

- Familiarity with the command line on Unix shells.
- Increase familiarity with phasers - OneDimAveraging.

Importants tips and links

edX site : https://edge.edx.org/courses/RiceX/COMP322/1T2014R

Piazza site : https://piazza.com/rice/spring2015/comp322/home

Java 8 Download : https://jdk8.java.net/download.html

Maven Download : http://maven.apache.org/download.cgi

IntelliJ IDEA : http://www.jetbrains.com/idea/download/

HJ-lib Jar File : http://www.cs.rice.edu/~vs3/hjlib/code/maven-repo/habanero-java-lib/hjlib-cooperative/0.1.4-SNAPSHOT/hjlib-cooperative-0.1.4-SNAPSHOT.jar

HJ-lib API Documentation : https://wiki.rice.edu/confluence/display/PARPROG/API+Documentation

HelloWorld Project : https://wiki.rice.edu/confluence/pages/viewpage.action?pageId=14433124

For Windows Users

Cygwin provides a standard UNIX/Linux shell environment, including many of its most useful commands to the Windows platform. It is free and open source software, released under the GNU General Public License version 3. Cygwin programs are installed by running Cygwin's "setup" program, which downloads the necessary program and feature package files from repositories on the Internet. You will need to install Cygwin using the instructions from the following web page: https://cygwin.com/install.html.

Lab Projects

The Maven project for this lab is located in the following svn repository:

- https://svn.rice.edu/r/comp322/turnin/S15/*NETID*/lab_7

Please use the subversion command-line client to checkout the project into appropriate directories locally. For example, you can use the following commands from a shell:

```
$ cd ~/comp322
$ svn checkout https://svn.rice.edu/r/comp322/turnin/S15/NETID/lab_7/ lab_7
```

1 Introduction to Basic Commands

This lab is designed to be a crash course in using the command line to make your computer perform tasks. You will be required to perform simple tasks to familiarize yourself with various commands.

Linux relies heavily on the abundance of command line tools. Most input lines entered at the shell prompt have three basic elements: command, options, and arguments. The command is the name of the program you are executing. It may be followed by one or more options (or switches) that modify what the command may do. Options usually start with one or two dashes, for example, -p or --print, in order to differentiate them from arguments, which represent what the command operates on. The interested student is encouraged to view Chapter 7 in the *LinuxFoundationX: LFS101x.2 Introduction to Linux* course on edX for a more thorough introduction on the command covered in lab today.

In order to help you organize your files, your file system contains special files called directories. A directory is a logical section of a file system used to hold files. Directories may also contain other directories. Directories are separated by a forward slash (/). The current directory, regardless of which directory it is, is represented by a single dot (.). The parent directory of the current directory, the directory one level up from the current directory, is represented by two dots (..). Your home directory is the directory you're placed in, by default, when you open a new terminal session. It has a special representation: a tilde (~).

You will experiment with the following commands:

1.1 pwd

`pwd` means "print working directory". It is used to output the path of the current working directory.

1.2 mkdir

`mkdir` means "make directory" and is used to create new directories. It creates the directory(ies), only if they do not already exist. Use the `-p` flag to ensure parent directories are created as needed.

1.3 cd

`cd` means "change directory". The `cd` command is used to change the current working directory. It can be used to change into a subdirectory, move back into the parent directory, move all the way back to the root directory, or move to any given directory. When the first character of a directory name is a slash, that denotes that the directory path begins in the root directory.

1.4 ls

`ls` means "list directory". The `ls` command lists out the contents of the directory you are currently in. You can use `cd` to change into different directories and then list what's in them so I know which directory to go to next.

1.5 dirs, pushd and popd

`pushd` means "push directory". `popd` means "pop directory". Both commands are used to work with the command line directory stack. The `pushd` command saves the current working directory in memory so it can be returned to at any time, optionally changing to a new directory. The `popd` command returns to the path at the top of the directory stack. This directory stack can be viewed by the command `dirs`.

1.6 rm

`rm` means "remove". Directories and files can be removed (deleted) with the `rm` command. By default, it does not remove directories. If the `-r` (`--recursive`) option is specified, however, rm will remove any matching directories and their contents. Use the `-f` (`--force`) option to never be prompted while files are being removed. The `-v` (`--verbose`) option can be used to get rm to detail successful removal actions.

1.7 cp

`cp` means "copy a file or directory". The command has three principal modes of operation, expressed by the types of arguments presented to the program for copying a file to another file, one or more files to a directory, or for copying entire directories to another directory. The commands takes two arguments, source and destination files, which may reside in different directories. You can use `cp` to copy entire directory structures from one place to another using the `-R` option to perform a recursive copy. Using this option copies all files, and all subdirectories from the source to the destination directory. you can specify multiple files as the source, and a directory name as the destination.

1.8 mv

`mv` means "move a file or directory". It moves one or more files or directories from one place to another, it is also used to rename files. When a filename is moved to an existing filename (in the same directory), the existing file is deleted. Use the `-f` (`--force`) option to never be prompted before overwriting existing files. The `-v` (`--verbose`) option can be used to get details on the actions and destination locations of the files being moved.

1.9 touch

The `touch` command is used to make an empty file. The `touch` command is the easiest way to create new, empty files. Touch eliminates the unnecessary steps of opening the file, saving the file, and closing the file again. It is also used to change the timestamps on existing files and directories.

1.10 less

It is used to view (but not change) the contents of a text file one screen at a time. Unlike most Unix text editors/viewers, `less` does not need to read the entire file before starting, resulting in faster load times with large files. You can open a file by passing the file name as an argument to the command. To traverse the file press the following, use the down arrow to scroll down one line while using the up arrow scrolls up one line. Using q will exit the less command.

1.11 cat

The `cat` command prints the contents of a file to screen and can be used to concatenate and list files. The name is an abbreviation of catenate, a synonym of concatenate. `cat` will concatenate (put together) the input files in the order given, and if no other commands are given, will print them on the screen as standard output. It can also be used to print the files into a new file as follows: `cat old1.txt old2.txt > newfile.txt` Typing the command cat followed by the output redirection operator and a file name on the same line, pressing ENTER to move to the next line, then typing some text and finally pressing ENTER again causes the text to be written to that file. The program is terminated and the normal command prompt is restored by pressing the CONTROL and d keys simultaneously.

1.12 find

The `find` command is a very useful and handy command to search for files from the command line. `find` will search any set of directories you specify for files that match the supplied search criteria. You can search for files by name, owner, group, type, permissions, date, and other criteria. The search is recursive in that it will search all subdirectories too. All arguments to find are optional, and there are defaults for all parts.

1.13 grep

The `grep` command is used to search text or searches the given file for lines containing a match to the given strings or words. Its name comes from the `ed` command `g/re/p` (globally search a regular expression and print), which has the same effect: doing a global search with the regular expression and printing all matching lines. By default, `grep` displays the matching lines. You can force `grep` to ignore word case with the `-i` option.

1.14 man

The man command is used to format and display the system's reference manuals. The man pages are a user manual; they provide extensive documentation about commands. Each argument given to man is normally the name of a program, utility or function. `man` is most commonly used without any options and with only one keyword. The keyword is the exact name of the command or other item for which information is desired, e.g. `man ls`.

2 Lab Task 1: Using the command line

1. Print the directory on STIC where your lab_7 project has been checked out.
2. Change the directory to `src/main/resources/commands`.
3. Print the contents of the current directory, you should be able to see three .txt files.
4. Use the `less` command to view the contents of the text files.
5. Create a directory called `work` in the current directory.
6. Use the `pushd` command and change to the `work` directory.
7. Use the `pwd` command to confirm you are inside the `work` directory.
8. Print the contents of the current directory, you should see no files.
9. Create an empty file called `loremipsum.txt` using the `touch` command.
10. Use the `cat` command to copy the text from `lorem.txt` and `lorem.txt` files in the `src/main/resources/commands` directory into the `loremipsum.txt` file.
11. Use the `mv` command to move the `loremipsum.txt` file into the `src/main/resources/commands` directory.
12. Print the contents of the current directory (`work`), you should see no files if the move has been successful.
13. Now use the `cp` command to copy all the `*.txt` files from the `commands` directory into the `work` directory.
14. Use the `popd` command to switch to the `commands` directory.
15. Delete all the files whose names start with `lor*` only in the current directory.
16. Use the `pushd` command and change to the `work` directory.
17. Use the `grep` command to find the `*.txt` files which contain the text `ma`.
18. Use the `find` command to find the files whose name contain the text `sum`.
19. Use the `find` command to find the files whose name contain the text `lor` and move them to the `src/main/resources/commands` directory.

3 Lab Task 2: One-Dimensional Iterative Averaging with Phasers

The code in `OneDimAveraging.java` performs the iterative averaging computation discussed in the lectures. This code performs a sequential version of the computation in method `runSequential`. **Your task is to create a more efficient phased version of `runSequential()` by using point-to-point synchronization with phasers instead.** Write this code in the space provided in `runPeerToPeer()`. The input arguments for the main method in this program are as follows:

1. `tasks` = number of chunks to be used for chunked parallelism. The default value for `tasks` is 8.
2. `n` = problem size. Iterative averaging is performed on a one-dimensional array of size `(n+2)` with elements 0 and `n+1` initialized to 0 and 1 respectively. The final value expected for each element i is $i/(n+1)$. The default value for n is 200,000.
3. `iterations` = number of iterations needed for convergence. The default value is 20,000. This default was set for expediency. For this synthetic problem, you typically many more iterations to guarantee convergence.
4. `rounds` = number of repetitions for the entire computation. As discussed earlier, these repetitions are needed for timing accuracy. The default value is 5. For 5 repetitions, a reasonable approach is to just report the minimum time observed.

You can run your programs locally by using the following maven command: `mvn clean compile exec:exec -POneDimAvg`. The command is also available in the README file in your project.

To run the program using 8 cores on STIC, use the `slurm` files and commands as described in previous labs. *WARNING: there is a lot of performance variability for the current implementation of phasers on STIC, so the timings that you obtain in this lab may be less repeatable than in previous labs.*

4 Turning in your lab work

For each lab, you will need to turn in your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab (as in COMP 215). Be prepared to explain the lab at a high level.
2. Check that all the work for today's lab is in the `lab_7` directory. If not, make a copy of any missing files/folders there. It's fine if you include more rather than fewer files — don't worry about cleaning up intermediate/temporary files.
3. Use the turn-in script to submit the `lab_7` directory to your turnin directory as explained in the first handout: *turnin comp322-S15:lab_7*. Note that you should *not* turn in a zip file.

   NOTE: Turnin should work for everyone now. If the turnin command does not work for you, please talk to a TA. As a last resort, you can create and email a `lab_7.zip` *file to comp322-staff@mailman.rice.edu.*