Lab 1: “Hello Linux!”

CPSC 1021 – Spring 2020

CLEMSON School of COMPUTING

Welcome to CPSC 1021 lab!

Every week we will meet on Tuesday and Thursday to complete a lab assignment. Lab assignment will be due on Friday of the same week before 5 pm. **I expect you to read the complete lab write-up each week**. Many questions about lab assignments can be answered by reading the material provided. With such short lab sections, we don’t want to waste time! So, if you ask your TAs a question that’s covered in the write-up, they will direct you back to the lab write-up. Let’s start!!!

Introduction

During this lab you will:

1. Login into the department Linux machines;
2. Practice using the command line interface;
3. Type, compile and run a small C++ program;
4. Create a zipped archive called “tarball”, and submit it to canvas.

Part I – Logging into Linux

To login into Linux you need your Clemson University username and password. Type them into the login boxes, and you are in!!! Now take a look around.

Linux is an operating system much like Windows and MacOS. An operating system’s job, broadly speaking, is to allow software applications to access hardware resources like the CPU and storage disks. Unlike proprietary operating systems (again, Windows/MacOS), Linux is notable in that it is open sourced and free!

Open source software is maintained and developed by the community at large, which means Linux was written by programmers for programmers. There are many flavors of Linux with varying degrees of user friendliness. In this lab we will use Ubuntu Linux, that has emerged as a great general-purpose OS.

Using your favorite search engine, please find out what the name “Ubuntu” means, and what language it comes from. Discuss your findings with the TAs.

Similar to Windows and MacOS operating systems, Linux has a graphical user interface (GUI, pronounced as “goo-ee”). But the real power of Linux is in its command line utility which allows users to quickly and efficiently type commands at the prompt without the distraction of the graphical elements. This is what we will use today and in the future labs.

Part II. Using command line interface

Creating directories

1. Right-click on the desktop and select Terminal. This will open a terminal window where you will type different commands. The terminal will usually open to your *home directory*.

2. Enter the *pwd* command. ("print working directory"). The working directory should be /home/*your_username*. This is known as your *home directory*. All your files and directories will be kept in a directory tree with this directory as root.

3. Enter *mkdir 1021labs* to create a directory named 1021labs for your labs this semester.

4. Enter *ls* to list the contents of the current directory. You should see the new *1021labs* directory.

5. Now you need to move into that directory. Enter *cd 1021labs* to change the working directory.

6. Enter *mkdir lab1* to create a folder for today's lab.

7. Enter *mkdir temp* to create a temporary directory. (you will learn how to remove this directory in a few minutes)

Using terminal editors to write code

1. There are several editors that you can use while writing code in the terminal. The easiest to use is *pico*. You can open it by typing *pico* on the command line. When it opens, you will see a window with a number of command abbreviations on the bottom. Press *Ctrl* and *O* (Overwrite means save) and give your file a name. Save your work very often, so you do not lose it! It does not take long to press *Ctrl* and *O* keys ans may save you a lot of headache later. To exit *pico* press Ctrl and *X* (X is for exit). There are also other options that you can explore on your own. This editor is very simple to use, though it does not have a lot of useful extensions.

2. Not all systems come with *pico*, but all of them come with *vim*. *vim* is a very powerful editor, but it does require a short learning curve to learn all the shortcuts. Knowing how to use vim is a valuable skill that each programmer should have. Next labs will be done using *vim*. In case you want to try *vim* today, here is a link with the beginner's guide: https://www.linux.com/tutorials/vim-101-beginners-guide-vim/. You can also google vim tutorials and youtube videos.

3. Alternatively, if you are familiar with Linux and already have your own favorite editor, you can use that editor today.

Using other commands

With the help of your TAs please practice using other commands:

cp *file1 file2* copies file named *file1* into a file named *file2*.

rm file1 deletes *file1.*

mv *file1 file2* renames *file1* to *file2.*

ls lists files and directories in the current directory

cd .. changes to a directory one level up.

cd ~ changes to your home directory.

clear clears the screen.

Getting additional help with Linux commands

Occasionally you will have questions about how a Linux command is used. There is no programmer who knows or remembers everything. It is very important that you can help yourself to find the necessary information. You have a couple of options here.

1. You can type the name of the command you need help with, followed by "--help", that is, two minus signs and the word *help* without spaces between them. For example, to know more about the command *cp*, you would type this at the prompt:

   cp --help | less

The | ("pipe") character and the word *less* after the command have a special purpose. The output of the command "cp –help" is "piped" or redirected into another command (*less*). *Less* is a Linux utility that formats the help entry so that if the entry is too long, you can see one screen at a time. It allows you to scroll up and down. To exit the help mode, press the *q* key on the keyboard. Try and run the command "cp –help" with and without the "| less" and observe the differences.

2. Another useful option is the online system manual, or *man*. Every Linux system comes with online documentation that explains how the system commands and utilities are to be used, and what options they have. To use the online manual to lookup a command, you would type:

man cp | less

As above, you will pipe your command into the *less* to be able to see information one screen at a time.

Now go ahead and practice using *man*. Look up such commands as less, more, cat, date, calendar, and clear. What happens if you type *man man* ?

3. Being able to use a search engine is also a good idea. Every time you come across something you do not know, you can help yourself right away by looking it up online. There is so much information out there waiting to be discovered! And the good news is, you already know how to look information up online.

Part III. Creating a simple C++ file, compiling and running it.

1. Using an editor of your choice, create a file named *lab1.cpp*. C++ files need to have an extension .cpp to let the compiler know that this is a C++ file. Type a small program that produces the output shown below. You can see examples in Chapter 2 of your textbook. Your output should be printed exactly as shown below, on two lines.

   This is my first C++ program !
   I love Linux !

2. Compile your program. The name of the compiler we will use is g++. Your executable file should be named *lab1.out*. To do this you will type the following command:

   g++ lab1.cpp -o lab1.out

   By default compiler would create an executable file named *a.out*. If we want to name our executable differently (for example, *lab1.out*), we need to specify this with the option "-o" at compile time.

   Why wouldn't we just leave the default name *a.out*? Imagine you have a number of files in the same directory, and you want to compile them all and keep their executables in the same directory. You cannot! Every time you compile a file, the newly generated *a.out* will overwrite the previously created one!

   Tu run your program just type **./lab1.out**, and you should see your output on the screen. If the output does not look like the one shown above, please go back and correct your code. Recompile and rerun.

Part IV. Creating a "tarball".

Before you do the next step, you need to backup the file you just created. Backing up a file means to create a copy of it in another location, so that if something went wrong, or you have accidentally deleted or corrupted your file, you still have a backup copy.

It is often necessary to create a zipped archive that contains several files. This is similar to zipping files on a Windows machine to create one .zip file. On a Linux machine this is called "creating a tarball", since the utility used to create it is called "tar", an abbreviation for "tape archiver". On older machines many years ago files were archived on a tape, and that name is still used for this useful utility. In this lab we will create a tarball with only one file in it – the C++

file you have created in the step above. The syntax of the command to create an archive is below:

tar cvf *file.tar dir*

The options “cvf” tell the *tar* utility what to do, for example, *c* means create an archive, *v* means do it in a verbose mode, *f* means file. *file.tar* is the name of your archive, it should have an extension “.tar”, and *dir* is the directory or the file you would like to archive.

NOTE: there is a possibility to corrupt/delete your work files if you do not use the command correctly. The name of the tar file should precede the name of the file/directory to archive.

After you have created an archive, you will use another utility to compress it.

gzip file.tar

This will compress the archive and now you will see an extension.gz added to the filename: *file.tar.gz.*

Extracting files from the tarball

To extract the files from the tarball you will have to unzip the file first by using the command:

gunzip file.tar.gz

This will unzip the archive and remove the extension .gz from the filename. Next you will extract files from the archive:

tar xvf file.tar

The option *x* means you are extracting files. Occasionally, you will want to examine the contents of the archive without actually extracting the files. You can do this with the following command.

tar tvf file.tar

t here is used to list the contents of the archive.

At this point you need to practice creating a zipped archive and extracting files from it. Make sure you have backed up your C++ file first, just in case.

NOTE: If you submit the archive that cannot be open, or is corrupt, you will not be given the opportunity to redo the work and resubmit. You have one shot to get it right.

What/How to submit

That’s it, you are done! You can now submit your zipped tarball to canvas to be graded.