COMP 2401 - Tutorial 1: Welcome to Linux and C

Learning Objectives

After this tutorial, you will be able to:

- Open and shut a virtual machine
- Navigate the file system on a Linux-based OS
- Consult the man pages and use the help command
- Create and edit text files
- Write and compile a simple C program using the GNU C compiler gcc
- Gain insight to some compiler warnings
- Learn how to use tar files

1 Create a SCS account (or re-new an existing one)

Log in as user 'newacct' and follow the instructions. This applies to those that already have accounts as they need to be re-created on the new Windows 10 setup.

2 Tutorial

2.1 Virtual Machine

Open Oracle VirtualBox and start the course's virtual machine: COMP2401-F19. You can log in with the following credentials:

username: `student`

password: `student`

The course's virtual machine is also available online at

https://carleton.ca/scs/tech-support/virtual-machines/

for installation on your personal devices.

To install the VM on your computer machine:

1. Download the course VM COMP2401_F19.ova to your computer
2. Make sure that you install the VirtualBox on your computer
3. Import the VM to VirtualBox
    a. Option 1
        - By clicking on the file COMP2401-F19.ova twice which most likely start the import process, or,

    b. Option B
        - start VirtualBox
        - Open the import appliance dialogue box (menu item file/import appliance)
        - Select the COMP2401-F19.ova file (you may need to navigate to the directory where you the *.ova file was downloaded to)

2.2 Introduction to Linux

It is essential that you are able to navigate the file system in a Linux/Unix system. Some commands such as '`ls`', '`cd`', '`cp`', and '`rm`' must be second nature to you.

2.2.1 Basic Unix Command

Once you have successfully logged in, open a shell (a command-line user-interface) by running the terminal program. You can do this by clicking on the *Terminal* icon in the task bar on the left-hand side of the screen.

In the shell, type the 'date' and 'pwd' commands, followed by the *Enter* key to execute them:

```
student@COMP2401-F19:~$ date
student@COMP2401-F19:~$ pwd
```

The first command is an example of a program external to the shell. The '`date`' command outputs the current date, while the '`pwd`' command is an example of a built-in shell command that prints the current working directory. You can identify which commands are external programs and which are built-in shell commands with the '`type`' command. Try the following:

```
student@COMP2401-F19:~$ type date
student@COMP2401-F19:~$ type pwd
```

To learn more about a command, you can use the 'man' command to read the manual page for an external command, or the 'help' command for a built-in command. Try reading the manual pages for the 'date' and 'pwd' commands:

```
student@COMP2401-F19:~$ man date
student@COMP2401-F19:~$ man pwd
```

The up and down curser keys will let you scroll through the manual pages. The space bar will move you down a whole page. Now try using 'help' for the previous commands:

```
student@COMP2401-F19:~$ help date
student@COMP2401-F19:~$ help pwd
```

2.2.2 Additional commands

Use 'man' or 'help' to read about each of the following commands: 'wget', 'echo', 'ls', 'cd', 'cp', 'rm', 'mv', 'mkdir', and 'rmdir'.

1. Listing files

Starting from the current directory, let's see what files and directories exist. First, try by simply typing 'ls' to list the files in the current working directory. You can modify how 'ls' displays information by giving it options. In BASH (the current shell in use, the '**B**ourne **A**gain **Sh**ell'), a convenient way to use 'ls' is with the 'a' and 'l' options; written as 'ls -al'. This is such a useful way of using 'ls', that there already exists an alias to the command; 'll' (ell ell). Using the 'type' command for 'll', you'll see that it's actually an alias and not a command. To see all the existing aliases, use the command 'alias'. Each line displayed is an alias that you can use.

You should have noticed that there are a lot more files (and directories) listed with 'ls -al' (ll) than with just 'ls'. The extra entries that start with a '.' are called hidden files/directories. Your home directory will usually have many of these, mainly due to configuration files. Type 'ls -l' to see all the non-hidden files and directories, and then read the manual page on 'ls' to see what the options mean.

2. Let's create a new directory:

```
student@COMP2401-F19:~$ mkdir testDirectory
```

Use the '`ls`' command to confirm that the new directory was created, and then move into this directory. You can change directory by using the '`cd`' command:

```
student@COMP2401-F19:~$ cd testDirectory
```

Note that as '`cd`' is a built-in shell command there is no manual page for it (though there is a help page). To know if you've actually moved into this directory you can always print the current working directory (`pwd`). Along with this, the prompt (where commands are entered) will display the current directory path.

What is in this directory? Well, there should only be two hidden directories: '`./`' and '`../`'. The first represents the current working directory and the second is its parent directory. This is extremely useful, since we can easily get back to the parent directory with '`cd ..`'.

We don't always have to remember the name of the parent directory. It is always accessible through '`..`'. If we want to move up two levels, to the parent directory of the parent directory, we could use '`cd ../../`'.

Now that we're done with this test directory, remove it.

3. Navigating the directories

Take some time to explore the file system. There isn't much in your home directory at the moment. Note that your home directory is not the main (root) directory of the file system. To navigate to this root directory, type '`cd /`'.

This changes the working directory to the root directory (`/`), which contains all files and directories on the system. Explore some of the file system starting from the root. In particular, notice that '`/bin`' and '`/usr/bin`' contain programs for the external commands you have been using so far (`mkdir`, etc). More information about the file system structure in Linux is located here.

Don't worry about trying to find your way back to your home directory. There are several ways that you can quickly do this. From anywhere in the file system, typing any of '`cd`', '`cd ~`', or '`cd $HOME`' will always bring you back to your home directory. The last example uses a shell variable, called '`HOME`', which stores the name of your home directory. To access variables, you need to use '`$`'. The tilde (~) is a built-in alias for '`$HOME`'. Try typing '`echo $HOME`'. This will print the value of the variable `HOME` to the screen (`/home/student`).

3 Using GNU compiler gcc - Creating a Hello World program.

3.1 Create the file

Use vi/vim (or the text editor of your choice) to write a hello world C program. To do this, begin by typing

```
student@COMP2401-F19:~$ vi hello.c
```

This launches vim. Hit the "i" key to enter the "insert" mode. Write the following code (exactly):

```
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return(0);
}
```

Hit <Esc> to enter the command mode (exit insert mode), then type ":wq" and hit enter to save the changes (write) and quit vi. For a more vim commands, see the file "basic_commands_for_vi.pdf" in this tutorial (file is also available in cuLearn) .

Or you can use emacs:

```
student@COMP2401-F19:~$ emacs hello.c&
```

3.2 Compile and execute the code

Next, compile your "hello.c" source code with the gcc compiler:

```
student@COMP2401-F19:~$ gcc hello.c
```

This command instructs gcc to compile the source code file "hello.c" and create an output file with the name a.out. The a.out file is an executable file.

Run your program by typing **a.out**. The program will most likely not execute because it was not found. The OS is searching for the executable file Type **./a.out** to run the program.

Tips:
The OS is using an environment variable called PATH to determine which directories to use. One can add the local directory to the PATH environment variable in order to avoid typing ./ every time one wants to execute a local executable such as a.out.
Steps:

1. Show the content of the environment variable PATH. Type from the command line echo $PATH
2. Add the local directory to the PATH environment variable. Type from the command line PATH=$PATH:.;
3. Check that the local path "." was added. Type from the command line echo $PATH It will show you all the directories that are included when the OS is searching for a program.

The file name a.out is a default executable file name that is used by gcc. If you are compiling and executing more than one c file in the same directory the executable a.out will be overwritten. In order to avoid it you can instruct gcc to give the output file a different file name.

Compile your "hello.c" again with the following gcc:

```
student@COMP2401-F19:~$ gcc -o hello hello.c
```

This command instructs gcc to compile the source code file "hello.c" and output the resulting program (machine code) to file the file "hello" (-o hello).

Recall: If you do not specify the output file name in gcc (by using the –o option), then it will by default use the file name "a.out".

Note, that at first you will compile your programs on the fly with a command line like the one above. Soon you will be introduced to make files (and the make program), which helps to automate the compilation process. However, before automating things, you need to understand what each option in the compilation line above means.

4 Saving hello.c on your Z drive (lab computers)

Open the file manager and locate your Z-drive. Copy hello.c to it and observe that you can now see it from the host computer (Windows 10 on the lab computers).

There is also a shortcut on the desktop of the virtual machine known as '`SharedFolders`', which bridges from the virtual machine to your Z-drive.

5 Fixing Warning messages

Warnings are hints from the compiler that something might not be right. They are not compile-time errors and the compiler will still create an executable machine language program if there are not actual errors (although, you can tell gcc to treat them like errors and not compile your code into machine code if a warning exists). It is always best to fix your code so that you do not have any warnings when you compile. Download **tw1.c, tw2.c, tw3.c and tw4.**c from the tutorial folder.

5.1 Fix the code in the files

5.1.1 Compile each of the programs tw1.c, tw2.c, tw3.c and tw4.c once with the –Wall flag and once without it. For instance:

```
gcc –o tw1 tw1.c

gcc –Wall –o tw1 tw1.c
```

5.1.2 Examine the warning and error messages that each compilation option has an output without warning or error messages.

5.1.3 Fix the programs so that no error or warning messages are produced.

5.1.4 Review the code to ensure that you understand what each program does

5.1.5 Execute the programs to ensure correct operation. If the program does not operate correctly, then fix it.

6 TAR Files

TAR files are files that were originally designed for archiving directories and creating backups on tapes. TAR stands for Tape Archive. In this course you will submit all your assignments using TAR files. In this part of the tutorial you will learn how to use TAR files.

1. From your home directory, create a new directory called '`Tutorial1`'.
2. Create a **Readme** text file containing

- your name and studentNumber
- a list of source files to be submitted (hello.c, tw1.c, tw2.c, ..)
- instructions for compiling and running the hello.c, tw1.c, tw2.c …

3. Move or copy hello.c and the four tw*.c files that you fixed into the '`Tutorial1`' directory.

4. Files with the '`.tar`' extension represent **T**ape **Ar**chive files, which are single files produced by the concatenation of their archived files. As such, the contained files are not compressed. These tar files are a common way of collecting files together into an archive in Unix/Linux.

The "tar" program manipulates tar files. As such it accepts command line options that instructs the program what to do

* To create a tar file – use –cvf to create it (-c is for creating the file, -v is for verbose and –f is for the target file name). The command is:

```
tar  –cvf   Tutorial1.tar  *
```

or

```
tar –cvf  Tutorial1.tar hello.c Readme tw1.c tw2.c tw3.c  tw4.c
```

* Listing what is in a tar:

```
tar -tvf Tutorial1.tar
```

* Extracting the files in a tar file – use –xvf (-x is for listing the content, -v is for verbose and –f is for specifying the target file name). The command is:

```
tar -xvf Tutorial1.tar
```

Additional options can be found in the manual page for the '`tar`' program.

5. Submit the '`Tutorial1.tar`' file on *cuLearn* under the Tutorial 1 section.

End of tutorial