- CPSC 316 PROJECT 0 -

- COMMAND-LINE PROGRAMMING IN THE LINUX ENVIRONMENT -

Most of your work for this class will be done in the Linux environment using command-line tools. The purpose of this assignment is to get some practice with the necessary account. We will perform a simple test of the development platforms we will be using in the class. Note that these have been tested on our lab machines. You may have to do something different if working from home.

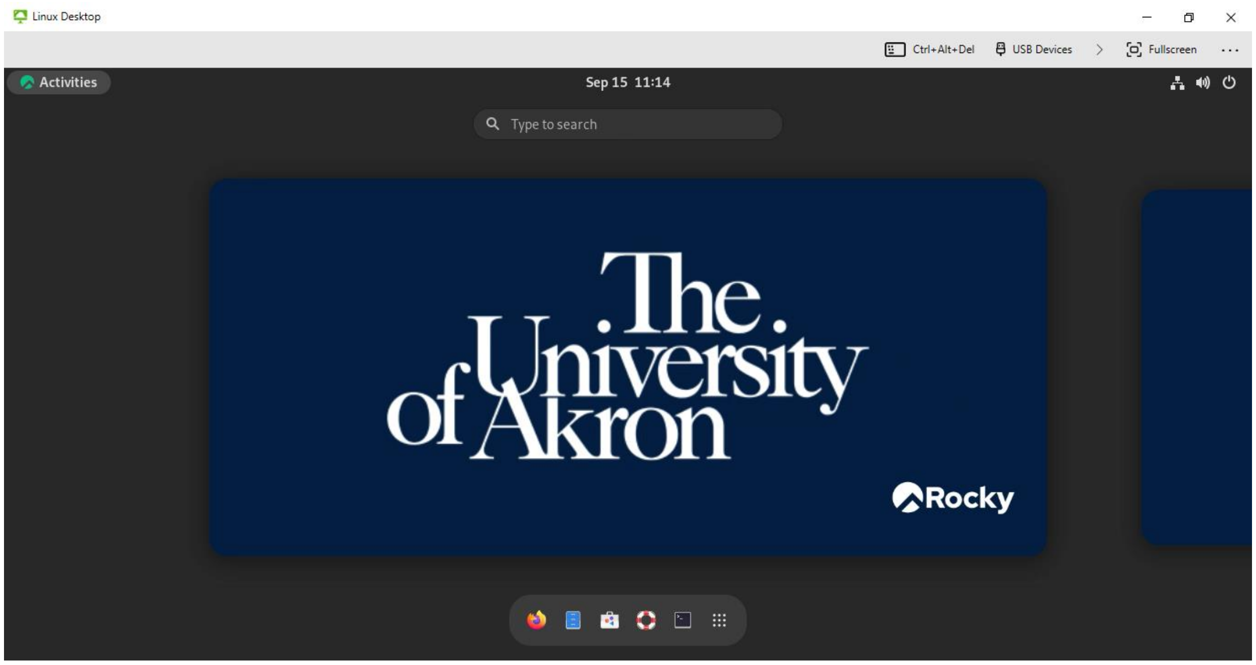
Initial Lab Setup

You will need access to the VMWare Horizon client. If working from our labs simply select the right icon to launch. If working from home, point your browser at *https://labs.uakron.edu* and "Click here to download VMWare Horizon client" at the bottom of the window. Follow instructions to install the native client.

Go to the course web page and download all files for the particular lab you're starting. For this exercise these include *hello.h*, *main.C* and *hello.C*. (The code from these appears in the Appendix to this document for your reference.) They should now appear in your **Downloads** folder.

Beginning Work

Launch the VMWare Horizon client, double-click on **labs.uakron.edu** and log in using your UA credentials. Double-click on **Linux Desktop** and log in again. Proceed to the Rocky Linux start-up screen pictured below. If you don't see this screen click on "Activities" in the upper left-hand corner.



At the top of this window click on **USB Devices** and make sure that both "Automatically Connect at Startup" and "Automatically Connect when Inserted" are checked. Note the six icons at the bottom of the window.

From left to right they are: **Firefox**, **Files**, **Software**, **Help**, **Terminal** and **Show Applications**. Finally take note of the "cycle power" icon in the upper far-right corner. Click this to select "Log Out" and terminate your session when appropriate.

Creating a Working Directory and Transferring Files

Once into your account select **Terminal**, then type **mkdir lab0** from the command line to create a folder in which to store your files for the rest of this lab. *Note that this is temporary storage that disappears when you end your session. Rember to save any files you work with to other media before logging out.*

From the terminal command line type **cd tsclient**, followed by **cd** *your user name* and then **cd Downloads**. There you should see the files you downloaded for this lab. To move them to **lab0** so you can work with them type **mv** *filename* **../../../lab0** for each.

From the terminal command line type **cd** then **cd lab0** to return to your lab work space. Finally type **ls -la**. You should see something similar to the screenshot below.

```
drwxr-xr-x  2 toneil domain users  66 Sep 15 11:47 .
drwx------ 16 toneil domain users 286 Sep 15 11:47 ..
-rw-r--r--  1 toneil domain users 201 Sep 13 15:16 hello.C
-rw-r--r--  1 toneil domain users  72 Sep 13 15:21 hello.h
-rw-r--r--  1 toneil domain users  74 Sep 13 15:18 main.C
```

The **ls** command is one of the most basic and essential commands in Linux. It's used to list the contents of the current directory. With these options it lists the complete content of the directory – including hidden files – in a long list format that includes detailed information.

- The first column shows the permissions for the file.
  - The first character is a dash (**-**) for a file, **d** for a directory and **l** for a link.
  - The second, third and fourth characters indicating read, write and execute permissions the owner has over the file.
  - Characters five through seven indicate permissions the group has over the file.
  - The remaining characters indicate permissions everybody else has over the file
- The second column shows the number of links to the file.
- The third column shows the owner of the file.
- The fourth and fifth column shows the group that owns the file.
- The sixth column shows the size of the file in bytes.
- The seventh through ninth columns show when the file was last modified.
- The tenth column shows the name of the file or directory.

Compiling Using the Linux Command Line

We now need to compile and execute our programs. The GNU C++ compiler is called g++. Typing

```
g++ hello.C main.C
```

will yield an executable called **a.out**. (You can see it is an executable if you do another **ls -la**. Observe that the 'x' flags are turned on for user, group and world.) Typing **./a.out** will now run our program. Once you've verified that this works delete this executable by typing **rm a.out**.

Using Makefiles

There are at least two problems with what we've done here. One issue is that the default name **a.out** is not descriptive and confusing to a novice programmer. Another is inefficiency. We are always recompiling every file, even if we only change one. To solve this will require multiple commands, which can be hard to remember and tedious to keep typing. This is addressed using a *makefile*. From your Linux command line type **nano makefile**. This command will start a simple text editor. Next, type each of these text lines.

Makefile

```
# Our simple makefile
hello: hello.o main.o
	g++ -g -o hello hello.o main.o
hello.o: hello.C hello.h
	g++ -g -c hello.C
main.o: main.C hello.h
	g++ -g -c main.C
clean:
	rm -f hello main.o hello.o
```

Very important: ***each of the indented lines must start with a tab character and not spaces***! Once finished typing type **Ctrl-O** and **Enter**, followed by **Ctrl-X** to save your file and exit the editor. Now type **make** at the command line. A quick check of **ls -la** reveals that our home directory has grown:

```
drwxr-xr-x  2 toneil domain users   108 Sep 15 12:04 .
drwx------ 16 toneil domain users   286 Sep 15 11:47 ..
-rwxr-xr-x  1 toneil domain users 43872 Sep 15 12:04 hello
-rw-r--r--  1 toneil domain users   201 Sep 13 15:16 hello.C
-rw-r--r--  1 toneil domain users    72 Sep 13 15:21 hello.h
-rw-r--r--  1 toneil domain users 32776 Sep 15 12:04 hello.o
-rw-r--r--  1 toneil domain users    74 Sep 13 15:18 main.C
-rw-r--r--  1 toneil domain users  3240 Sep 15 12:04 main.o
-rw-r--r--  1 toneil domain users   195 Sep 15 12:04 makefile
```

This time our executable is named the more descriptive *hello*. Type **./hello** at the command line to run the program. Next type **make clean**, then **ls**. You see that all the extra files have been deleted and we're back where we started from.

Let's now back up and review. First of all, comments in a makefile start with a hashtag ('**#**') and continue to the end of the line. The makefile contains mostly rules, with each rule containing a *target*, *dependencies* and then *commands*. The target is the name of a file; dependencies are files needed to create the target. For example, to create the **hello** executable, I need the files **hello.o** and **main.o**. If those files don't exist, I execute the rules that create those files before coming back to this one.

But what of the actual commands executed (seen at right)? In all commands the **-g** flag adds debugging information to the executable that may be useful later. The **-c** flag creates object files (**hello.o** and **main.o**) that contain machine code and will be linked together to form the executable. Lastly, the **-o** flag changes the executable name.

```
g++ -g -c hello.C
g++ -g -c main.C
g++ -g -o hello hello.o main.o
```

Finally, we often include a target called *clean* that removes all the object files and executables. By default, **make** builds the first target in the makefile. By specifying **clean** as the argument to **make** we execute this rule, which simply executes a forceful removal (**rm -f**) of the named files. In other words the files are deleted without you being prompted for confirmation.

Compressing Directories and Saving Your Work

Creating a ZIP file compresses one or more files or folders into a single file, which keeps your work organized and easy to submit. Type **cd ..** to back out of your **lab0** directory, then **zip -r** *yourUserID* **lab0**. This archives everything into one file with your name on it that can be easily attached to email or submitted to an online drop box.

As said before, all of your work will be deleted once you terminate this Linux session. Your best bet is saving everything to a flash drive. Plug your drive in and **cd tsclient** from the terminal prompt. There a capital letter (most likely **D**) will be added along side your UA ID. Save your work via the command **mv ../lab0/ D/**. Type **mv tsclient/D/lab0/ .** from your Linux prompt to restore it from your flash drive the next time you log in. Obviously you can do the same with your zipped final version to put it somewhere you can upload it from.

Conclusion

This exercise was intended as practice for the work we will be doing the rest of the term. As such there is nothing to turn in. Just be sure you understand everything well enough to be able to complete future assignments.

Appendix: Source Code

hello.h
```
#ifndef HELLO_H
#define HELLO_H
void hello( );
void shutUp( int );
#endif
```

main.C
```
#include "hello.h"
int main( ) {
  hello( );
  shutUp( 3 );
  return 0;
}
```

hello.C
```
#include <iostream>
#include "hello.h"
using namespace std;
void hello( ) {
  cout << "Hello world!" << endl;
}
void shutUp( int n ) {
  for( int i = 0; i < n; ++i )
    cout << "Shut up!" << endl;
}
```

Last updated 9.15.2023 by T. O'Neil. Previous revision 2.24.2023.