- CPSC 316 PROJECT 0 -

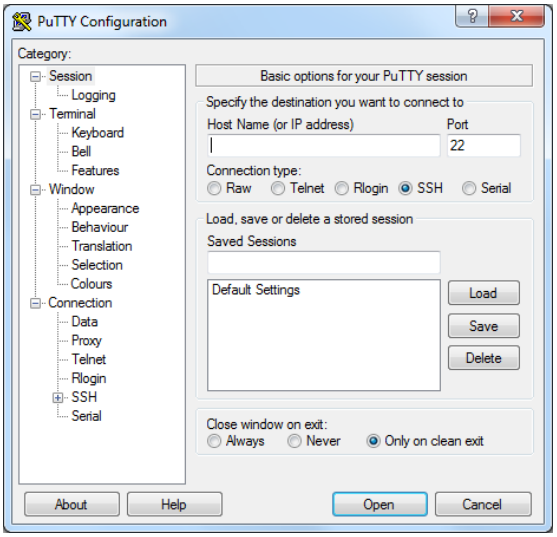
- COMMAND-LINE PROGRAMMING IN THE LINUX ENVIRONMENT -

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 set up the necessary account. We will also perform a simple test of one 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.

Setting Up Your Department Account

Using the guest password and account, log into the department's system (*https://fs1.cs.uakron.edu*; you either need to be on campus or working across a VPN) and set up your c.s. account for the semester. Once activated, this account will be used to access the knuth server.

PuTTY is the recommended application to use for SSH connections from a Windows operating system. PuTTY allows you to access your files and email stored on our servers. It also provides a UNIX environment to run programs that some courses require. PuTTY can be used to do this from any Windows computer, on or off campus.



Once you have the software, launch PuTTY and you will see the screen at right. In the "Host Name (or IP address)" field, type: "knuth.cs.uakron.edu" and select **[Open]**. Log in with your CS account information. Once into your account type **mkdir lab0** from the command line to create a folder in which to store your files for the rest of this lab.

File Transfers Using WinSCP

From the class web page download the files *hello.h*, *main.C* and *hello.C* to your "My Documents" folder. (The code from these appears in the Appendix to this document for your reference.) We will now copy these files to knuth by using *winscp* from your Windows account. This software is installed on our lab machines for your use. You will have to download and install it (from *https://winscp.net/eng/download.php*) if working from your own computer.

WinSCP (Windows Secure Copy) is an open source SecureFTP client for Windows. It allows secure file transfers between the client's local computer and the remote server. When opened, the program will automatically open a new **Login** window where you will be prompted to provide login information about the location you want to connect to. In that window fill in the following:

- **File Protocol** – choose **SFTP** from the drop-down;
- **Host Name** – knuth.cs.uakron.edu;
- **Port number** – 22;
- **User name** – your c.s. account username;
- **Password** – your c.s. account password.

When ready press **[Login]** to connect to your account. You will be prompted to add the identity of the target server to the cache, click **[Yes]** to not see this warning in the future.

You are now connected to your account and can see all files and folders in it on the right-side panel. On the left-side panel, you can find all the files and folders on your computer. If you want to use WinSCP to upload files or folders on your hosting account, you just need to navigate to the desired file or folder on the left side panel and drag-and-drop it to the right-side panel. Similarly, it is very easy to also download data from your hosting account to your machine. Navigate to the location of the file or folder you want to download from the right-side panel and drag-and-drop it to the left-side panel.

For this lab, navigate to the local computer's "My Documents" folder in the left-side panel. There you should see the three files downloaded earlier. On the right-side panel double-click on the **lab0** directory you made earlier. Drag-and-drop the three files for this lab into your knuth workspace. When finished, use PuTTy to log back into knuth, type **cd lab0** then **ls -la**. You should see something similar to the screenshot below.

```
drwxr-xr-x  2 toneil users 4096 Feb 18 17:35 .
drwx------ 52 toneil users 4096 Jan 31 14:21 ..
-rw-r--r--  1 toneil users  201 Feb 18 17:35 hello.C
-rw-r--r--  1 toneil users   72 Feb 18 17:33 hello.h
-rw-r--r--  1 toneil users   74 Feb 18 17:34 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 column shows the group that owns the file.
- The fifth column shows the size of the file in bytes.
- The sixth column showed when the file was last modified.
- The seventh 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.

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 either **pico makefile** or **nano makefile**. Either 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 users  4096 Feb 18 20:14 .
drwx------ 52 toneil users  4096 Jan 31 14:21 ..
-rwxr-xr-x  1 toneil users 29424 Feb 18 20:14 hello
-rw-r--r--  1 toneil users   201 Feb 18 17:35 hello.C
-rw-r--r--  1 toneil users    72 Feb 18 17:33 hello.h
-rw-r--r--  1 toneil users 37832 Feb 18 20:14 hello.o
-rw-r--r--  1 toneil users    74 Feb 18 17:34 main.C
-rw-r--r--  1 toneil users  2800 Feb 18 20:14 main.o
-rw-r--r--  1 toneil users   195 Feb 18 20:14 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

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.

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 2.24.2023 by T. O'Neil.