**Lab 1: Introduction to Unix Systems and C**

**Objectives**

By the end of today's lab, you will be able to

* Login to and navigate the School of "Computing's Unix workstations
* Create subdirectories, list directory files, and execute other Unix commands
* Use one of the available text editors to edit a simple program
* Compile and execute a simple C program

**Resources**

1. [UNIX Command Basics](https://www.cs.clemson.edu/course/cpsc1070/reference/unix.html)
2. [Basic vi Commands](https://www.cs.clemson.edu/course/cpsc1070/reference/vi-commands.pdf)
3. [Software download page](https://ccit.clemson.edu/services/software-hardware/software/web-downloads/)

**Procedures for each lab**

For each lab you will:

1. Create a new directory for the lab
2. Review the overview for the lab to get a feel for its objectives and what you will do during the lab.
3. Use *lab1071copy* to copy any documents that are provided for the lab.
4. Submit the completed work for grading as directed in the instructions.

**Task 1 -- Directory Operations and Other Unix Commands**

For task 1 carry out the following steps:

1. On either the Linux virtual machine on your laptop or a lab workstation, log into the system.
2. Once you are logged in open up a terminal window (you can right-click on the desktop and select "Open Terminal").
3. Enter the ***pwd*** command. ***pwd*** stands for "print working directory". The workding directory should be either ***/users/your\_username*** or ***/home/your\_username***. This is also known as your *home directory*. Terminal windows open to this location by default. All your files and directories will be kept in a directory tree with this directory as root.
4. Create a directory in your home folder by typing ***mkdir lab1071***
5. Enter ***ls*** to list the contents of the current directory. You should see the new lab1071 directory that you just created.
6. Enter ***cd lab1071*** to move to the lab1071 directory.
7. Within the lab1071 directory, type ***mkdir lab1*** to create a new folder for lab1.
8. Enter ***ls*** to list the contents of the current directory. You should see the new lab1 directory that you just created.
9. Enter ***cd lab1*** to move to the lab1 directory.   Now you can jump back to your lab1071 directory by typing ***cd ../***.  From the lab1071 directory, you can jump back to your home directory by typing ***cd ../***.  ( you could have gone directly from your lab1 directory to your home directory by typing ***cd ../..***.  each ".." jumps back one level. Note that the command line prompt shows you which directory you are currently in. The tilde "~" indicates your "home" directory.) You can also always return to your home directory by simply typing ***cd***.
10. More Unix Commands: You should practice using each of these additional commands.

|  |  |
| --- | --- |
| **Command** | **Description** |
| rm <filename> | Remove (i.e., delete) file named <filename> |
| rmdir <name> | Remove (i.e., delete) empty directory named <name> |
| mv <source> <target> | Move <source> to <target> |
| pwd | Print the working directory to the screen |
| cd <directory> | Change to <directory>. If <directory> isn't specified change to your home directory. |
| ls <directory> | List contents of directory. Use '-l' flag to see long view. Omit <directory> to list contents of current directory. |
| less <filename> | Show contents of file on screen, enabling you to scroll back and forth |
| cat <filename> | Show contents of file on screen |
| exit | Close the terminal |
| control-c | Sends the terminate signal to a running process (i.e., kill the current process). |
| man <command> | Display the manual page (i.e., help) for <command>, containing lots of help-type information. |

**Task 2 -- Editing**

We will now create the "hello world" program in your "lab1" subdirectory.

1. If you are in your home directory you can return to the lab1 directory by entering ***cd lab1071/lab1***.
2. Next we want to invoke an editor to create the program. There are a number of editors that are available under Unix. Serious Unix users usually use editors like "vi", "vim", "gedit", or "emacs". "vi" (pronounced "v eye") is the instructor's favorite and is described in Chapter 8 of the "Unix in a Nutshell" book ("vim" is an enhanced "vi" editor that is generally available on a Unix system).

If you are doing your work on a local machine like the virtual machine on your laptop or a lab machine (i.e. you are not connecting to the machine remotely over the network) "gedit" is a popular choice and is similar in use to NotePad under Windows. If you don't know any other Unix editor go ahead and enter the command:

***gedit hello.c***

In gedit you can use the mouse to maneuver around, and you can highlight code, use the delete button, etc. to edit your text. **Note: you can use gedit in the labs or on your Linux virtual machine but you CANNOT use gedit over a Secure Shell (ssh, remote) connection. You will have to use an editor like nano, pico, vim or emacs.**

Other simple editors include "nano" (or "pico") which a number of Clemson students use (simply enter the command "nano").

**Nano**

The "nano" editor is an easy-to-use choice that can be used over an "ssh" network connection to a remote machine (e.g. if you are connecting from your laptop to a department lab machine). To create a file a new file in the current directory, type

**nano <filename>**

nano will open for editing the given file. If the file already exists, it will open, if not, a new file with the name will be created. Navigate nano in the following manner:

* + The arrow keys move the cursor around
  + ctrl-d deletes
  + ctrl-e finds the end of a line
  + ctrl-k cuts a line of text
  + ctrl-u uncuts a line of text (pastes)
  + ctrl-o saves the file
  + ctrl-x exits nano

Now, using the editor of your choice, open the file "hello.c" and enter the following code:

/\*

Ima Coder

file name: hello.c

<today's date goes here>

Description: This program displays "Hello, world!"

\*/

#include <stdio.h>

#include <stdlib.h>

int main()

{

printf("Hello world\n");

return 0;

}

When you have finished entering and editing this (perhaps you've figured out that you can cut and paste if you wish), then save the file. In gedit click the "File" tab, and then hit "save". In nano enter "Ctrl-X" (hold down the Ctrl key and the X key simultaneously), and then enter "Y", and finally the "Enter" key to save the file.

1. Now enter the "**ls**" command. You should see the file "hello.c" listed as being in the current directory.
2. If you are using gedit you will find that it is important that the editor display line numbers for the source code. Enter gedit again, i.e.

**gedit hello.c**

Now look to the left of the source code lines. If you see line numbers then you are okay, and you can just exit gedit.

If you do not see line numbers then click on the "Edit" tab on the top line of the gedit window. Then click on preferences. This should bring up a menu of options. Check the "Display line numbers" box, and "Close". You should now see line numbers to the left of the source code lines. Now exit gedit.

**Task 3 -- Compiling**

Next we want to compile the hello program. We do this by entering the command:

**gcc -g -Wall hello.c**

Hopefully the program compiled without error (if you have typos, you will see error messages. If the program is syntaxtically correct, you'll only see the Unix prompt again.) If you do have errors, you will need to re-edit the program and correct the error.

**Task 4 -- Executing**

Now we are ready to run the program. First do another **ls** command. You should now see the file "a.out" in addition to "hello.c". The file "a.out" is the *default* name given to the file created by gcc unless we specify otherwise (more on this later).

Now to execute the program, type:

**./a.out**

You should see

**Hello world**

If you get something different, you should fix the program and try again.

If you want to rename the executable to something other than **a.out**, then use

**gcc -g -Wall -o newname hello.c**

and to execute:

**./newname**

**Task 5: Modify the Program**

1. Use your favorite editor again to modify your ***Hello World*** to print the following output:

**Hello world  
from Clemson University   
Home of the Tigers!**

1. Compile and execute your program to make sure that it still works.

**Task 6: Formatting and Comments**

1. Always indent your code in a readable way. Your lab instructor will discuss proper indenting for this course.
2. The following comments should be inserted at the beginning of the file:
   * Header comments, which include
     + Your name
     + File name
     + Date
     + A brief description of the program, indicating what the program does
3. There should be a comment before each function, except *main*, that
   * Describes the function
   * Identifies the function parameters

Now, insert header comments (your name, file name, date, description) in the ***Hello World*** program, then compile and test the program again to make sure that it still works.

**Task 7: Compile Errors**

The GNU C Compiler program (gcc) is an open-source C compiler released by the Free Software Foundation's [GNU Project](http://www.gnu.org/) (where GNU is a recursive acronym ofr GNU's Not UNIX). GNU was started as an attempt to create a free clone of the commercial UNIX systems of the day, and the project members created a number of free operating system utilities and compilers, including gcc. GNU never quite finished its own operating system kernel (the HURD kernel). Instead, its tools are used on commercial systems, such as Solaris and Mac OS C, and an open-source Linux and \*BSD systems. Each of these distinct systems was derived from, or a clone of, the original Blll Labs UNIX system. Due to trademark considerations, this large class of operating systems is often called \*NIX.

In CPSC 1070/1071, we will be using the gcc compiler on a CentOS version of Linux. The command to run gcc is (mostly) the same, however, across other \*NIX systems, including Sun Solaris, Mac OS X, and FreeBSD.

To invoke the gcc compiler, use this command:

gcc -o executable-file -Wall -g file1.c file2.c ...

In CPSC 1070/1071, we treat errors and warnings as having the same severity, since warnings often indicate that the program will behave in a buggy or unsafe way. (There are exceptions, with warnings that are harmless, but we will not encounter these cases here.) The -Wall turns on additional warnings that can help catch bugs. We will always use this flag in 1070/1071.

When gcc compiles a program successfully, there is not output: only the next shell prompt appears and the executable file is created. If there is an error (a condition so severe that no program can be created), gcc does not create the executable file, and an error message is printed. If only warnings occur, the filename you specify after the "-o" parameter gets created, and the warnings are also printed.

When gcc does emit errors or warnings, such messages consist of four componenet:

1. The name of the source file in which the problem occurred
2. The line number on which the problem was detected
3. The word "warning" or "error"
4. gcc's description of the (potential) problem

For instance, here is possible output from running gcc on broken.c:

user$ gcc -o broken -Wall -g broken.c

broken.c:6: warning: return type defaults to 'int'

broken.c: In function 'main':

broken.c:10: warning: implicit declaration of function 'scanf'

broken.c:10: warning: incompatible implicit declaration of built-in function 'scanf'

broken.c:10: warning: format '%d' expects type 'int \*', but argument 2 has type 'int'

broken.c:10: warning: format '%d' expects type 'int \*', but argument 3 has type 'int'

broken.c:10: error: invalid lvalue in assignment

broken.c:10: error: expected statement before ')' token

broken.c:12: warning: implicit declaration of function 'printf'

broken.c:12: warning: incompatible implicit declaration of built-in function 'printf'

broken.c:12: error: missing terminating " character

broken.c:13: error: expected ')' before ';' token

broken.c:14: error: stray '\' in program

broken.c:15: error: incompatible type for argument 1 of 'printf'

broken.c:15: error: expected ';' before '}' token

broken.c:15: error: expected ';' before '}' token

broken.c:17: warning: control reaches end of non-void function

When a program has many problems, gcc can issue messages that are misleading with respect to both the cause and location of problems. Often, fixing the topmost problem will fix several of the errors and warnings below it. However, until you become comfortable with the C programming language and gcc, **try compiling after every 5-10 lines of code**, once you have the skeleton of the program written.

**Task 7 Exercise**

1. If you are not already there, change back to your "~/lab1071/lab1" directory,
2. **Copy files**

For this and future labs, you will use the "lab1071copy" to copy any necessary files. First, type the below into your terminal:

gedit ~/.bashrc

Now add the following line to the bottom of the file, save, and exit gedit.

alias lab1071copy="/group/course/cpsc1070/public\_html/labs/lab1071copy.sh"

In order for the alias to take effect, you will either need to restart your terminal or you can type the below

source ~/.bashrc

Now that you have the copy command setup, you will not need to set it up again in the future. You can now use it to download your lab files. The syntax of the command is as follows

lab1071copy <lab #> [directory]

Where lab # is simply the number of the lab you would like to download the files for and directory is the directory the files will be copied to. If you leave off the directory argument the script will put the files in your current directory.

For example, to copy the files for this week's lab (which is lab 1) simply type

lab1071copy 1

You should see the below output

copying files...

/group/course/cpsc1070/public\_html/labs/lab01/Public/broken.c

/group/course/cpsc1070/public\_html/labs/lab01/Public/data.txt

--copy complete--

1. Use the gcc command (with -Wall, -g and -o broken) to compile the source file, *broken.c*. Fix the first warning or error, try compiling again, fix the next warning or error, and so forth until you get a *clean compile*, i.e. one that produces the executable file and does not produce any errors or warnings.
2. Once you have an executable file, execute the program as follows:

./broken < data.txt

**Task 8: Submission**

Use [Handin](https://handin.cs.clemson.edu/) to submit ***hello.c*** and ***broken.c***. Handin is a Web-based handin application located at: https://handin.cs.clemson.edu

**Note:**It is your responsibility to make sure that assignments are correctly submitted. Check your handin folder after submitting.