Fall 2024
CS3413
Lab1
Bash scripting
Due Wednesday 11 Sep 5:00 PM Fredericton time.

Mac users, you're likely to use CMD instead CTRL throughout the lab if you are working on your own device.

This lab is to introduce the class to Linux CLI and Terminal.

Your task is to find out about the commands that can help you create files and directories, navigate directories, check various statistics of the lab equipment, and submit the script you created in process, as well as your activities log. All the activities are to be executed using command line terminal (once you launch it your way). Please login into the lab machine your preferred way. Open the terminal program.

Remote access instructions (extract from Course Overview on D2L)

To complete this course students will need to have access to a Linux machine running a standard C programming development environment including: text editor, compiler and pthreads library. Students can access the UNB FCS labs via vpn/vnc or they can run an FCS linux image via virtualbox.

- For instructions on remotely connecting to FCS's labs, you can find directions here:
    - GUI access (https://www.cs.unb.ca/help/remote-lab-gui-access.shtml)
    - Command line access (https://www.cs.unb.ca/help/ssh-help.shtml)

BEFORE STARTING WORKING ON THIS LAB

Read about the 'history' Linux command. Once you log in, please type \`history\` to observe the history of commands you've previously executed.

For this assignment, you will be asked to submit the output of your history command as well as the script you've created. To avoid the TA's or the instructor learning something you don't want to share, please delete your previous history (don't forget to back it up if you need it (see submission instructions to learn how to back up the history)

You should execute \`history -c\` before starting to work on this lab. Please work within the same terminal window.

First and foremost:

Google can be helpful in figuring out the commands you need to use, for sure. However, if you find yourself in the situation when you're not allowed to use internet, you can use Linux manuals to investigate the command. You are allowed to use internet for this lab. Also please ask your TA or instructor for help.

Working on this lab:

Make sure you typed **history -c** before starting to work on this lab

Use **ls** to examine the contents of the folder you're in.

Use **pwd** to identify the current working directory.

Use **mkdir "Lab 1"** to create a directory called "Lab 1". Note you will need to either escape the space, like Lab\ 1 or use quotes.

Type **cd "Lab 1"** to change your directory to Lab 1.

Confirm the current directory has changed using one of the previous commands.

Notice that the "Lab 1" name is lame because it makes you use quotes.

Type **cd ..** to go one directory higher.

Confirm the current directory has changed using one of the previous commands.

Confirm the annoying "Lab 1" folder is still there by using one of the previous commands.

Delete the folder using **rm "Lab 1"**

Learn that "Lab 1" is a directory and shell won't delete it for you. Use **rm -r "Lab 1"** to delete the directory with a space in its name

Create a new directory "lab1".

Realize your instructor asked you to create a folder with a lowercase "l" letter in its name to make you practice renaming files and directories. Type **mv lab1 Lab1** to rename the folder.

Confirm the folder was renamed and start typing **cd D** and press Tab on your keyboard multiple times to learn there's an autocomplete and you don't always need to type all names yourself.

Notice that **cd D** and hitting **Tab** has nothing to do with your Lab1 folder, press **ctrl+c** combination to cancel the current command.

Start typing **cd La** , hit Tab and enter Lab1 directory.

Verify you're where you're supposed to be.

Create a file test.c by using **touch test.c** command

If you're familiar with how to use emacs or vi or vim properly, your instructor applauds you and invites you to use the shortcuts that you're comfortable with. For the rest of the class

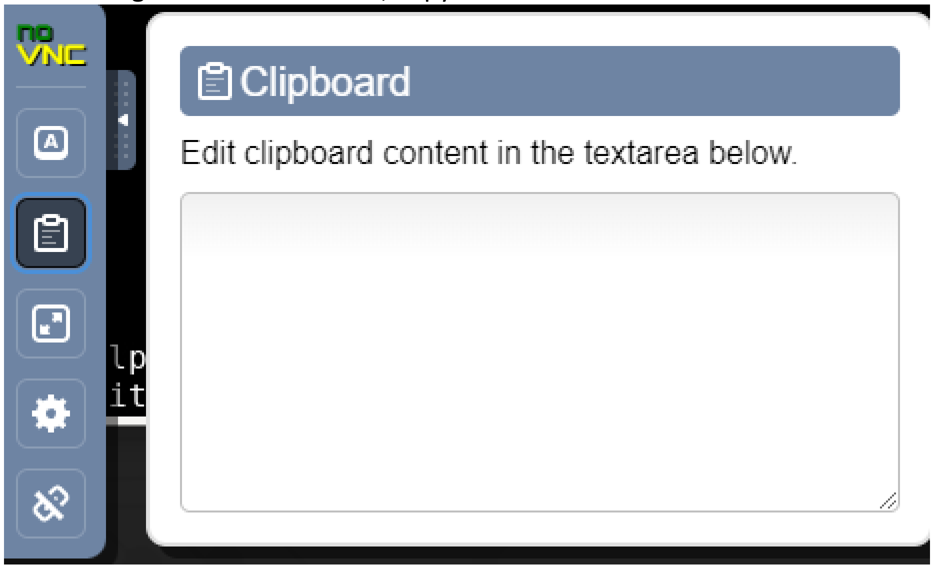
Type **nano test.c** and observe text editor. Think about how great it is to have a GUI text editors. Notice Nano provides you with hints about its interface.

The lab continues on the next page!

Select the following lines and copy them by using **ctrl+insert**

```
#include<stdio.h>
int main(int argc, char** argv){
char temp[100];
gets(temp);
printf("%s\n",temp);
return 0;
}
```

If you are using the GUI access to the lab pc, use the clipboard to insert the text there when you are transferring the data. After that, copy the code from within the VM.



Shift+insert the lines into your nano editor.

Press **F2** (or **ctrl+x**) combo to start writing to the file.

Hit **Enter**

Answer **Y** to the prompt.

Now, type **gcc -Wall -Werror test.c**

Notice how compiler does not let you to proceed with using deprecated function, that's due to -Wall (show all warnings) and -Werror (treat all warnings as errors).

Replace the problematic line with scanf. Look up the appropriate use of this function

Recompile your program.

The **echo** command allows to print output to screen. The **>** token allows redirecting the output of your command to a text file. Try it by typing **echo Hello world > input.txt**

Notice that single **>** token will create input.txt whereas **>>** will append to existing contents.

Try it by typing **echo Hello planet >> input.txt**

Verify the contents of the file by using **cat input.txt**

Now type **echo Hello > correct_output.txt**

Please, also type **echo Hello world > incorrect_output.txt**

The **<** token allows redirecting contents of a file to serve as a standard input. We've covered that in class a little, but now your instructor wants you to try it yourselves. Please do that by typing **./a.out < input.txt**

Notice that you have your output on the screen. Now, you can combine the tokens together! Try **./a.out < input.txt > output.txt**

Verify the contents of the file by using **cat** command.

Now you want to verify if the correct output matches exactly the output your program produced. Type **diff correct_output.txt output.txt**

If nothing is displayed on your screen – that's exactly what diff generates if the files are identical. Try verifying how **diff** works if the files differ: press up arrow for your **diff correct_output.txt output.txt** to appear on the screen. Modify this command to look like **diff correct_output.txt incorrect_output.txt** and observe the outputs. You can do that by using backspace or delete and arrow keys for navigation. Note how diff does not care about the file extensions matching.

Pipe symbol **|** allows connecting output of one program to input of the second program.

tail -n # program allows you to list the number **#** of lines from reading a file. Try calling the history command and limiting its output to only include 20 last lines. Try redirecting the results into a file called **script.sh** using **>** token

Assuming that you have created script.sh file, you need to edit the file only to contain the lines related to

1. compiling your file
2. running the program with input/output redirection
3. comparing the output and correct output.

If your file does not have the related lines, you can call **history | tail** to output more lines. Please remove the line numbers and time stamps, leaving just the commands. Save the file and exit the **nano** program.

Create a file called **tmp.txt** using the **nano** program. To do that, type **nano tmp.txt**

Modify the file contents, so that

#!/bin/bash

is the first and only line in the file. This line, if placed into a **.sh** file allows making sure that BASH interpreter is used (there several possible shells like ksh, zsh and other).

cat command was originally designed to concatenate files, not just observe the file contents. Try merging contents of **tmp.txt** and **script.sh** Try doing that by **cat tmp.txt script.sh** and redirecting the output into a file that is called **compile_script.sh**

Try executing your file by calling **./compile_script.sh** . Notice that your shell says that permission is denied.

Try changing the fille permissions to make it executable. Try adding (**+**) permissions to e**x**ecute this file by typing **chmod +x compile_script.sh** .

Try executing the file again. Your script should compile, run, and compare the output of a c program. Notice you now have a script like how your instructor and TAs will be grading your program. Feel free to verify your programs before submitting them to D2L.

Please proceed to submit your work.

Submission instructions:

Type

history > history_backup.txt

This set of commands can be used to back up your history of commands;

cat history_backup.txt compile_script.sh > to_submit.txt

Please check if your **to_submit.txt** contains history of your previous commands and clean unrelated commands before submitting.

Please submit the **to_submit.txt** file.