Wayne State University

CSC – 4421 Computer Operating Systems Lab
Winter 2020
Lab 0 – Introduction to Linux

Instructors
Rui Chen – Section 001

Goals

The purpose of this lab session is to help you get introduced to the structure and requirements of this class and get familiar with GNU/Linux in the following aspects:

- Use the help/manual feature;
- Edit & compile C/C++ code;
- Use pointers in C;
- Understand the homework submission procedure.

Lab submission process

Please read the following instructinos carefully. Solutions submitted in other formats or using other procedures will NOT be accepted.

- There are several tasks in each lab. Only tasks with an asterisk need to be submitted. Other tasks will not be counted into your grade but are required.

Collaboration Policy

This is an individual assignment, as will be all assignments in this course. You may help each other verbally with the correct execution of GNU/Linux commands. You may not type anything in another student's session, nor provide any code or answers to the questions below.

Tasks

1. Start using Ubuntu to complete the lab assignment.
   1.1. Start Ubuntu
   1.2. Start a terminal session
   1.3. List directories
      - ls -l
   1.4. Navigate to the Desktop
      - cd Desktop
   1.5. Navigate up one level in the directory tree
      - cd ..
   1.6. Navigate back to the Desktop

- cd Desktop

1.7. Create a folder for this class; make a folder for this lab; navigate to the folder for this lab
- mkdir csc4421
- mkdir csc4421/Lab00
- cd csc4421/Lab00

2. Try out the following GNU/Linux commands in the session you started in Task 1 (press enter after each one).
   - who
   - whoami
   - uname
   - ls
   - mkdir Folder1
   - cd Folder1
   - ls
   - cat /etc/os-release
   - cp /etc/os-release
   - ls
   - cat os-release
   - mv os-release newname.txt
   - ls
   - rm newname.txt
   - ls
   - cd ../
   - ls
   - rmdir Folder1 (or) rm -r Folder1
   - ls
   - ls -l

3. There are a number of editors for Linux. Learn to use one of them. Here are a links to a few of them: (gedit is default application in Ubuntu)
   3.1. gedit: http://projects.gnome.org/gedit/
   3.2. nano: http://debianadmin.com/nano-editor-tutorials.html
   3.3. emacs: http://www.gnu.org/software/emacs/tour/
   3.4. vim: http://www.vim.org/docs.php

4. To enable administrator rights, use sudo.
   To install you can use apt-get. Type “sudo apt-get update; sudo apt-get install aptitude”
   Now you can use aptitude instead of apt-get.
   - note: “apt-get update”: updates the list pf available packages. Usually, it is recommended to update the list of packages, before installing a new one.

   Choose an editor. Try vim, emacs, pico, nano, gedit, or anything you like. Create a new text file with the editor of your choice named testfile.txt and add several lines of text to it.

   Type the following commands

cat testfile.txt
less testfile.txt
echo something
echo something > redirect.txt
history
history >> redirect.txt
ls –a ~/
less ~/.bash_history

Try pressing up and down to see your previous commands. Try typing “cat re” and then press tab. Press Ctrl+r and start typing “echo so” and you will see the history command that you typed.

To copy and paste to and from the clipboard in the terminal use Ctrl+Shift+c and Ctrl+Shift+v.
- note: you have to add the shift key

5. The GNU/Linux **man** command is an extremely useful way to learn more about GNU/Linux commands (and also system calls).

   Type the command:

   man man

   to learn a little about the **man** command. It is probably more than you wanted to know. Keep hitting the space bar to get the next screen. Look for the following sections as you go:

   - NAME
   - SYNOPSIS
   - AVAILABILITY
   - DESCRIPTION

6. Were there any commands that you had trouble with in Task 2? Use the **man** command to loop up some of those commands (at least **3**).

7. Now take a look at the man page for **ls**. Find the answers to the following questions and write them up.
   (a) What is the default soft order for **ls**?
   (b) Is “R” a legal option for **ls**?
   (c) Is “q” a legal option for **ls**?
   (d) What does “**ls -a**” do?
   (e) In what directory can “**ls**” usually be found? (The command “ls” is a file, which executes a program to do the **ls** command.)
   (f) What other **man** pages might you want to read for more information?

   Put your answer in the file called “csc4421_accessId_Lab00_task7.txt” in folder “csc4421_accessId_Lab00_task6” in the root folder “csc4421_accessId_firstName_lastName_Lab00”

8. *(X points) Input the program below, using one of the GNU/Linux editors. Name our program "Command-Line.c"

```c
#include <limits.h>
#include <stdio.h>
#include <unistd.h>
#ifndef PATH_MAX
#define PATH_MAX 255
#endif
int main(void) {
    char mycwd[PATH_MAX];
    if (getcwd(mycwd, PATH_MAX) == NULL) {
    perror("Failed to get current working directory");
    return 1;
    }
    printf("Current working directory: %s\n", mycwd);
    return 0;
}
```

Compile the program input in Task 8 using gcc or g++.

Compile the program inputted in Task 8 using.

gcc Command-Line.c

This will create an executable called a.out. You can name the executable something else with -o as we do in task 9.

Run it to make sure it works. To run an executable file you need to write:

./ [executable]

In this case it is:

./a.out

What does getcwd do?

9. Create a makefile

9.1. Create a file name **makefile** – it must be called **makefile** – and input the following into that file (and save):

```
Command-Line: Command-Line.c
        gcc Command-Line.c -o Command-Line.out
```

9.2. In the Terminal window, use the **make** command followed by **ls:**

- **make ls**

You will see a new executable called Command-Line.out

Type:

./Command-Line.out

To test it.

10. *(Y points) Input the program below, using one of the Linux editors. Name your program “pointer.c”

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

void
f(void)
{
    int a[4];
    int *b = malloc(16);
    int *c;
    int i;

    printf("1: a = %p, b = %p, c = %p\n", a, b, c);

    c = a;
    for (i = 0; i < 4; i++)
    a[i] = 100 + i;
    c[0] = 200;
    printf("2: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n",
       a[0], a[1], a[2], a[3]);

    c[1] = 300;
    *(c + 2) = 301;
    3[c] = 302;
    printf("3: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n",
       a[0], a[1], a[2], a[3]);

    c = c + 1;
    *c = 400;
    printf("4: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n",
       a[0], a[1], a[2], a[3]);

    c = (int *) ((char *) c + 1);
    *c = 500;
    printf("5: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n",
       a[0], a[1], a[2], a[3]);

    b = (int *) a + 1;
    c = (int *) ((char *) a + 1);
    printf("6: a = %p, b = %p, c = %p\n", a, b, c);
}

int
main(int ac, char **av)
{
    f();
    return 0;
}
```

Compile the program input in Task 7 using gcc or g++.

(a) Run it to make sure it works.
(b) Understand the output.
(c) Change only the line "c = (int *) ((char *) a + 1);" of the code so that you can change line 5 of output to be "5: a[0] = 200, a[1] = 400, a[2] = 500, a[3] = 302
(d) Upload source code and screen shot of the result.

Put the modified code named "pointer-modified.txt" in folder

"csc4421_accessId_Lab00_task8-pointer-modified.c" in folder
"csc4421_accessId_Lab00_task8" in the root folder
"csc4421_accessId_firstName_lastName_Lab00"

11. To zip…

zip [archive_name].zip [file/directory_to_compress] (use -r for a directory)

Notice that your current directory must be outside the folder to be compressed in order to run the command correctly.

To unzip…

unzip [zipname].zip

12. *(z points) Compress your assignment folder "csc4421_accessId_firstName_lastName_Lab00" and upload it to blackboard.

zip csc4421_accessId_firstName_lastName_Lab00.zip
csc4421_accessId_firstName_lastName_Lab00/

Note: the above is one line, it is one two here to fit on this page.