

Written by: Dr. Farkhana Muchtar

# LAB 1: BASIC LINUX COMMANDS

## Introduction of Linux OS

## 1 Overview

These exercises aim to acquaint students with the Linux command-line environment and introduce them to some essential Linux commands, with a focus on those specific to Ubuntu or Debian distributions.

### (a) If you are installing Linux using guidelines in Lab 0:

To access Linux instance, use **ssh** command in your **Windows Terminal**. Make sure to start Linux virtual machine with headless mode :

```
ssh <username>@192.168.56.10
```

You advisable to use **Powershell 7** in **Windows Terminal**.

### (b) If you are installing Ubuntu/Debian/Kali Desktop Linux in virtual machine based on the Youtube video tutorial:

Just open Terminal application

## 2 Simple Linux Command

Linux commands are executed on Terminal by pressing **Enter** key at the end of the line. You can run commands to perform various tasks, from package installation to user management and file manipulation.

Here's what a Linux command's general syntax looks like:

```
CommandName [option(s)] [parameter(s)]
```

A command may contain an option or a parameter. In some cases, it can still run without them. These are the three most common parts of a command:

- **CommandName** is the rule that you want to perform.
- **Option** or **flag** modifies a command's operation. To invoke it, use hyphens (-) or double hyphens (--).

Parameter or argument specifies any necessary information for the command. Keep in mind that all Linux commands are case-sensitive.

# SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

In this lab exercise, you will learn and practice essential Linux commands. Follow the step-by-step instructions to complete each task.

Before we start, run this two command as preparation to our lab exercise

```
mkdir share; touch example.txt share/textfile01.txt
```

## Task 1: Navigating directories

1. Use the `pwd` command to display the current working directory. **pwd** is acronym for “**P**rint **W**orking **D**irectory”.

```
muhammad@secr2043:~$ pwd
/home/muhammad
muhammad@secr2043:~$ |
```

2. Use `ls` to list down contents of current directory. In your case, it might be an empty directory.

```
muhammad@secr2043:~$ ls
example.txt  share
muhammad@secr2043:~$ |
```

We can list down contents of other directory.

```
ls <directory>
```

For the example, we want to know what are inside directory `share`. Simply run this command

```
muhammad@secr2043:~$ ls
example.txt  share
muhammad@secr2043:~$ ls share
textfile01.txt
muhammad@secr2043:~$ |
```

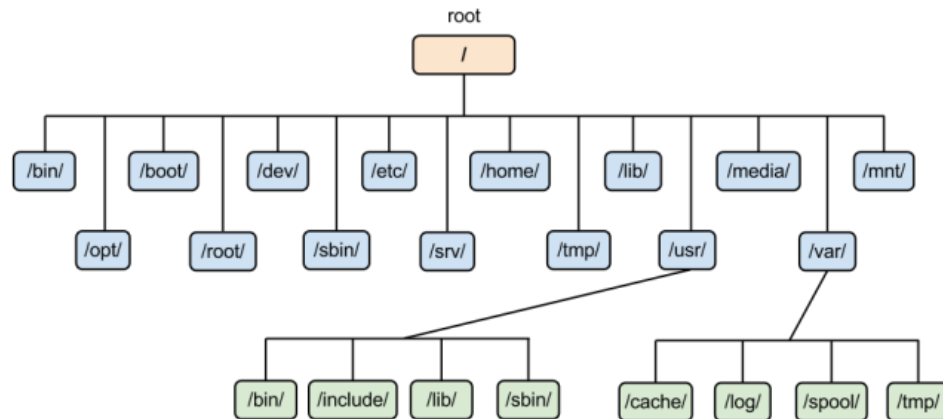
Seems like `share` directory are empty. How about we list down another directory, which is upper level directory. We list down the `/` directory, the highest level of directory.

```
muhammad@secr2043:~$ ls /
bin          dev          lib          lost+found  opt         run          snap      tmp
bin.usr-is-merged  etc         lib.usr-is-merged  media      proc       sbin        srv       usr
boot        home        lib64        mnt        root      sbin.usr-is-merged  sys       var
muhammad@secr2043:~$ |
```

# SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

In Linux and other Unix-like OS, the file system is hierarchical and `/` is the root of all directory.



The other way to list down upper level directory is by using `..` (double dot), which mean one level upper than current directory.

For example, currently, you are inside `/home/<username>`. When you run this command,

```
ls ..
```

The output is

```
muhammad@secr2043:~$ ls ..
farkhana mahyuddin muhammad student
muhammad@secr2043:~$ |
```

What it shows is list of directory inside `/home`, which is directory that is one level above `/home/muhammad`. We also can use `..` (double dot) more than one time to retrieve upper level directory like this

```
ls ../../..
```

and the output is

```
muhammad@secr2043:~$ ls ..
farkhana mahyuddin muhammad student
muhammad@secr2043:~$ ls ../../..
bin          dev          lib          lost+found  opt          run          snap        tmp
bin.usr-is-merged etc        lib.usr-is-merged media        proc         sbin         srv         usr
boot         home        lib64        mnt          root         sbin.usr-is-merged sys         var
muhammad@secr2043:~$ |
```

In this case, two times double dot means, two upper level directory of `/home/<username>`, which is `/`.

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

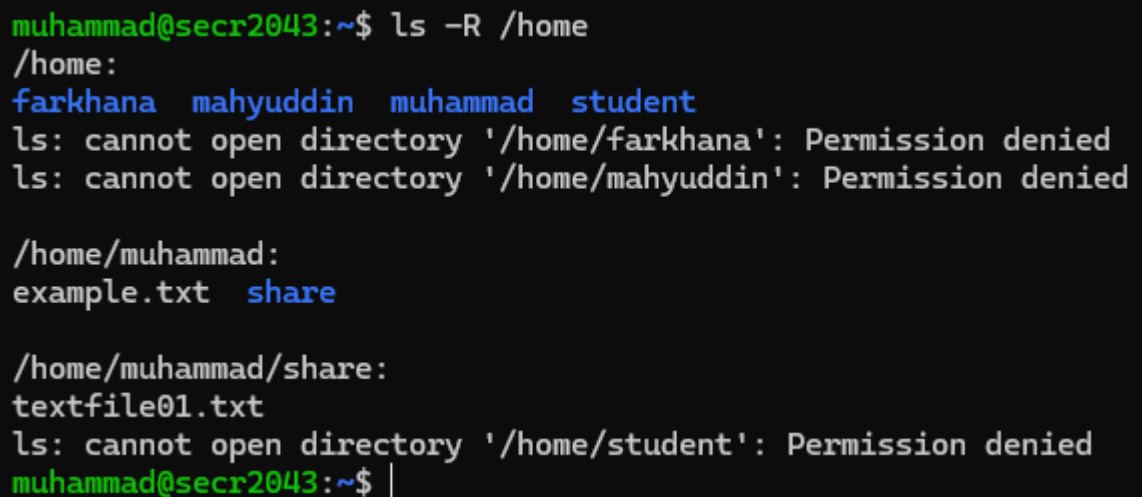
However, for this exercise, we only focusing on directory `/home/<username>`, which is home directory for your username. We will dig down further about Linux file system in the next lab exercise.

Here are some options you can use with the `ls` command:

- `ls -R` lists all the files in the subdirectories.
- `ls -a` shows hidden files in addition to the visible ones.
- `ls -l` stands for long format. It shows Unix file types, number of hard links, permissions, group, owner, last modified name and date-time, and size.
- `ls -lh` shows the file sizes in easily readable formats, such as MB, GB, and TB.

Example for `ls -R`:

```
ls -R /home
```



```
muhammad@secr2043:~$ ls -R /home
/home:
farkhana mahyuddin muhammad student
ls: cannot open directory '/home/farkhana': Permission denied
ls: cannot open directory '/home/mahyuddin': Permission denied

/home/muhammad:
example.txt share

/home/muhammad/share:
textfile01.txt
ls: cannot open directory '/home/student': Permission denied
muhammad@secr2043:~$ |
```

As you can see here,

- The `/home` directory contains subdirectories: `farkhana`, `mahyuddin`, `muhammad`, and `student`.
- For `farkhana`, `mahyuddin`, and `student`, the command fails with the **'Permission denied'** error. This indicates that the user `muhammad` does not have permission to access these directories.
- For `/home/muhammad`, the command succeeds, listing two items: `example.txt` and a subdirectory `share`.
- Inside `/home/muhammad/share`, there is a file named `textfile01.txt`.

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

Example for `ls -a` :

```
ls -a ~
```

```
muhammad@secr2043:~$ ls -a ~
.      .bash_history  .bashrc  .profile      example.txt
..     .bash_logout  .cache   .sudo_as_admin_successful  share
muhammad@secr2043:~$ |
```

- The `ls -a` command lists all files and directories, including hidden ones (those starting with a dot, like `.bash_history`).
- Hidden files:  
`.bash_history`, `.bash_logout`, `.bashrc`, `.cache`, `.profile`, `.sudo_as_admin_successful`
- The `~` symbol represents the user's home directory, which for user **muhammad** is `/home/muhammad`.

Let's use `-l` option and `-lh` options to list down files and directory inside user home.

```
muhammad@secr2043:~$ ls -l
total 4
-rw-r--r-- 1 muhammad muhammad 0 Apr 7 20:12 example.txt
drwxr-xr-x 2 muhammad muhammad 4096 Apr 7 20:12 share
muhammad@secr2043:~$ ls -lh
total 4.0K
-rw-r--r-- 1 muhammad muhammad 0 Apr 7 20:12 example.txt
drwxr-xr-x 2 muhammad muhammad 4.0K Apr 7 20:12 share
muhammad@secr2043:~$ |
```

Both are showing details information of each file and directory. The only difference between `-l` and `-lh` is, `-lh` shows file and directory size in human readable format.

3. Create a new directory called `linux-lab` using the `mkdir` command:

```
mkdir linux-lab
```

```
muhammad@secr2043:~$ ls
example.txt  share
muhammad@secr2043:~$ mkdir linux-lab
muhammad@secr2043:~$ ls
example.txt  linux-lab  share
muhammad@secr2043:~$ |
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

4. Create another directory called workspace

```
mkdir workspace
```

```
muhammad@secr2043:~$ ls
example.txt  linux-lab  share
muhammad@secr2043:~$ mkdir workspace
muhammad@secr2043:~$ ls
example.txt  linux-lab  share  workspace
muhammad@secr2043:~$
```

5. Change the current working directory to linux-lab using `cd linux-lab` command and using `pwd` to display current working directory.

```
pwd
```

```
muhammad@secr2043:~$ cd linux-lab
muhammad@secr2043:~/linux-lab$ pwd
/home/muhammad/linux-lab
muhammad@secr2043:~/linux-lab$ |
```

6. Use the `cd` command to navigate to your home directory.

```
muhammad@secr2043:~/linux-lab$ cd
muhammad@secr2043:~$ pwd
/home/muhammad
muhammad@secr2043:~$
```

7. Run `rmdir <empty_directory_name>` to delete empty directory. For this tutorial, run `rmdir linux-lab`.

```
muhammad@secr2043:~$ ls
example.txt  linux-lab  share  workspace
muhammad@secr2043:~$ rmdir linux-lab
muhammad@secr2043:~$ ls
example.txt  share  workspace
muhammad@secr2043:~$ |
```

\* Remake linux-lab directory again by typing `mkdir linux-dir` for next exercise.

```
muhammad@secr2043:~$ ls
example.txt  share  workspace
muhammad@secr2043:~$ mkdir linux-lab
muhammad@secr2043:~$
```

# SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

## 3 Simple Text File Manipulation

We already learned some text manipulation commands in previous section and will explore in more details in this section.

*Make sure you are inside the **linux-lab** directory (created above) before starting these steps.*

### Task 1: Working with files

1. **Create an empty file:** Use the touch command to create an empty file named **file1.txt**:

```
touch file1.txt
```

Running `ls` now should show **file1.txt** in the directory.

2. **Write text to a file using output redirection:** Use the echo command to write some text into **file1.txt**, redirecting the output into the file:

```
echo "Hello, world" > file1.txt
```

Here, the `>` symbol **redirects** the output of the echo command into the file **file1.txt** (creating the file if it didn't exist, or overwriting it if it did).

3. **View the contents of the file:** Use the cat command to display the contents of **file1.txt**:

```
cat file1.txt
```

You should see the text **Hello, world** printed to the terminal.

4. **Overwrite the file with new text:** Use echo with the `>` redirection again to put different text into **file1.txt**:

```
echo "Welcome to Operating System lab exercise" > file1.txt
```

This will replace the previous content of **file1.txt** with the new text.

5. **Verify the overwrite:** Run `cat file1.txt` again. Now it should show **Welcome to Operating System lab exercise**, and the original "Hello, world" line will be gone. (*Using `>` overwrote the file's content.*)

6. **Append text to the file:** To add new text to the file without removing the existing content, use the append operator `>>`:

```
echo "This is basic Linux command exercise." >> file1.txt
```

The `>>` operator appends the output to the end of the file.

7. **Verify the append operation:** Run `cat file1.txt` once more. You should see that **file1.txt** now contains both the previous line **Welcome to Operating System lab exercise** and the new line **This is basic Linux command exercise**. The second line was added to the file instead of overwriting it.

- **Note:** The `>` operator redirects output **and overwrites** the target file, while `>>` **appends** the output to the end of the file (or creates the file if it doesn't exist).

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

8. **Capture command output in a file:** You can use redirection with other commands as well. For example, use `ls -l` to get a long listing of the directory and redirect that output to a file **filelist.txt**:

```
ls -l >> filelist.txt
```

Since **filelist.txt** did not exist, this command creates it and places the output of `ls -l` into it. (We used `>>` here; using `>` would have the same result in this case, but `>>` ensures that if you run the command multiple times, it would append each new output to the file instead of overwriting.)

9. **Display the new file's contents:** Use `cat filelist.txt` to see what was recorded. You should see the long-format listing of the **linux-lab** directory (which includes **file1.txt**, **filelist.txt** itself, etc., with details like permissions, owner, size, and modification date).
10. **Rename a file:** Use the `mv` (move) command to rename **file1.txt** to **file2.txt**:

```
mv file1.txt file2.txt
```

Now, if you run `ls`, you will see **file2.txt** (instead of **file1.txt**). The file has been renamed.

11. **Copy a file:** Use the `cp` command to make a copy of **file2.txt** and name the copy **file3.txt**:

```
cp file2.txt file3.txt
```

Running `ls` again should show that you now have **file2.txt** and **file3.txt** in the directory (along with **filelist.txt**).

12. **Concatenate two files (display combined output):** The `cat` command (short for "concatenate") can take multiple files and print their contents one after another. To view the contents of **file2.txt** followed immediately by **file3.txt**, run:

```
cat file2.txt file3.txt
```

You will see the text of **file2.txt** followed by the text of **file3.txt** in the terminal output. (At the moment, **file3.txt** is an exact copy of **file2.txt**, so you'll see the same content twice in a row.)

13. **Concatenate two files (save combined output to a new file):** You can also redirect the combined output into a new file. For example:

```
cat file2.txt file3.txt > combined.txt
```

This reads the contents of **file2.txt** and **file3.txt**, concatenates them, and writes the result into a new file **combined.txt**.

**Verify the new combined file:** Run `ls` to confirm **combined.txt** was created. Then use `cat combined.txt` to view its contents. It should contain the content of **file2.txt** followed by the content of **file3.txt**. (Again, if those two files were identical, **combined.txt** will just have that content repeated.)



# SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

## Task 3: Working with file permissions

Every file in Linux has permission settings that control who can read, write, or execute it. Let's inspect and modify the permissions of **file2.txt**.

1. **View file permissions:** Use a long listing to see file permissions:

```
ls -l file2.txt
```

This will output a line that starts with something like `-rw-rw-r--`. This string represents the permissions for the file (and some other information). For example, `-rw-rw-r--` breaks down as:

- `-` (file type, `-` means a regular file)
- `rw-` (owner permissions: here the owner can read and write)
- `rw-` (group permissions: members of the file's group can read and write)
- `r--` (others permissions: everyone else can only read)

The owner of the file is typically you (your user account), and by default, you should have read and write permission.

2. **Change file permissions:** Use the `chmod` command to remove the write permission for the owner of **file2.txt** (simulating making the file read-only for yourself):

```
chmod u-w file2.txt
```

Here, `u` means "user/owner", `-w` means "remove write permission". This command will update the file's permissions so that the owner can no longer edit the file.

3. **Verify the permission change:** Run `ls -l file2.txt` again. Now the permission string at the start should show `-r--rw-r--` (or similar), indicating that the owner no longer has the "w" (write) permission.

*Note:* Without write permission, if you try to edit or append to **file2.txt** now, you will get a "Permission denied" error. The file is effectively read-only for you (until you change the permissions back or use `sudo` to override, which we won't do now).

4. **(Optional) Restore write permission:** For the purposes of continuing to work with the file, you may restore your write access using:

```
chmod u+w file2.txt
```

3. Running `ls -l file2.txt` after this would show the original permissions (owner has `rw` again). This step is optional here because we will not need to modify the file further, but it's good to reset the permission if you plan to edit or use the file normally later.

## Task 4: Searching for files and content

Now that we have multiple files with content, you can practice searching for text within files and searching for files in directories.

1. **Search within a file using grep:** The `grep` command looks for occurrences of a string within files. For example, let's search **file2.txt** for the word "Welcome" (which we know is in the file from earlier steps). Run:

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

```
grep "Welcome" file2.txt
```

This will output any line in **file2.txt** that contains the word "Welcome". If the word isn't found, it will output nothing. You can try searching for other words that you know exist in the file (for instance, try a word you added in Nano).

*Tip:* `grep` is case-sensitive by default. You can add the `-i` option to make it case-insensitive, or use other options like `-n` to show line numbers. For example, `grep -n "linux" file2.txt` would search for "linux" and prefix each matching line with its line number.

2. **Search for a file by name using find:** The `find` command can search for files in a directory hierarchy. For example, to find the file named **file3.txt** in the current directory (and any subdirectories), run:

```
find . -name file3.txt
```

Here, `.` means "start searching in the current directory", and `-name file3.txt` tells `find` to look for files with the name "file3.txt". This should locate our **file3.txt** and print `./file3.txt` (the `./` indicates it's in the current directory).

You can use `find` to search by other criteria as well (by size, by modification date, etc.), but name is the simplest and most common use in daily tasks. For example, `find / -name "*.log"` would search the entire system for files ending in `.log` (though that might be slow and require root permissions for some directories).

### Task 5: Cleanup

Now we'll clean up the files and directories we created during the lab.

1. **Remove a file:** Use the `rm` command to delete **file3.txt** (since this was just a copy, we can remove it as a cleanup step):

```
rm file3.txt
```

This will permanently delete **file3.txt**. (Be careful with `rm` – it does **not** move files to a trash/recycle bin; it removes them immediately.)

2. **Navigate out of the directory:** We are about to remove the entire **linux-lab** directory, so first move out of it to your home directory:

```
cd ~
```

(You can replace `~` with `../..` or an absolute path if you prefer, but `~` is quick for home.)

3. **Remove a directory with all its contents:** Use `rm -r` (recursive remove) to delete the **linux-lab** directory *and everything inside it*:

```
rm -r linux-lab
```

This will remove **linux-lab** and all the files we created in it (such as `file2.txt`, `combined.txt`, `filelist.txt`, etc., if they still exist). You may be prompted to confirm deletion for each file or to confirm the recursive deletion (depending on your system's settings or aliases). If prompted, confirm each removal.

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

4. **Verify removal:** You can run `ls` to check that **linux-lab** is gone. If you also want to clean up the **kongsi** directory and **example.txt** we created in the preparation step (and if you no longer need them), you can remove those as well:

```
rm -r share
rm example.txt
```

*(Make sure you're in the directory that contains those, such as your home directory, before running these.)*

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 4 Using Nano Text Editor

In this section, you will learn the basics of the Nano text editor, a user-friendly command-line tool for creating and editing text files in Linux. By the end of this exercise, you will be able to:

- Open Nano and create a new file.
- Edit text and navigate within the editor.
- Save changes and exit Nano.
- Use basic shortcut keys for efficiency.

Nano is a simple, easy-to-use text editor for Unix-like operating systems, including Linux. It is designed to be a user-friendly alternative to other command-line text editors, such as Vi or Emacs.

Nano is especially useful for beginners who are not yet familiar with the more complex features and commands of other text editors.

#### 4.1 Launching Nano

To open Nano, type the following command in your terminal and press Enter:

```
nano
```

This opens a blank editor where you can start typing.

**Tip:** You can open Nano with a specific filename to create or edit a file. For example:

```
nano myfile.txt
```

This creates a new file named `myfile.txt` (or opens it if it exists).

#### 4.2 Creating a New File

If you open Nano without a filename, you can name the file when saving.

For this exercise, create a file directly by running:

```
nano hello.txt
```

#### 4.3 Editing Text

Once Nano is open, start typing. Use the arrow keys to move the cursor.

For example, type:

```
Hello, World!
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 4.4 Saving Changes

To save your work, press `Ctrl + O` (Write Out).

Nano will prompt you to confirm the filename. Press **Enter** to save.

**Note:** If you didn't specify a filename when opening Nano, it will ask you to enter one here.

### 4.5 Exiting Nano

To exit, press `Ctrl + X`.

If you have unsaved changes, Nano will ask if you want to save them (type Y for yes or N for no).

### 4.6 Basic Navigation

Use these commands to move around in Nano:

- `Arrow keys`: Move the cursor left, right, up, or down.
- `Ctrl + A`: Jump to the beginning of the current line.
- `Ctrl + E`: Jump to the end of the current line.
- `Ctrl + Y`: Scroll up (previous page).
- `Ctrl + V`: Scroll down (next page).

### 4.7 Useful Shortcut Keys

Nano shows shortcuts at the bottom of the screen (the ^ symbol means Ctrl). Here are some key ones:

- `Ctrl + G`: Open the help menu.
- `Ctrl + W`: Search for text in the file.
- `Ctrl + K`: Cut the current line.
- `Ctrl + U`: Paste the cut text.

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 4.8 Nano Exercises

In this exercise, you will learn how to use the Nano text editor to create, edit, and manipulate text files in Linux. You'll practice opening files, saving changes, exiting Nano, and using shortcuts - all with a provided sample text to make the process easier.

#### 4.8.1 Sample Text for Exercises

This text is designed for the exercises below and includes multiple lines and keywords like "Linux" for searching.

```
Linux is an open-source operating system.  
It powers servers, desktops, and even small devices.  
Linux offers great stability and security.  
Popular versions include Ubuntu, Fedora, and Debian.  
Learning Linux is a valuable skill for tech enthusiasts.  
Nano is a text editor that runs in the Linux terminal.  
It's easy to use, even if you're new to Linux.  
With Nano, you can edit files or write code.  
Linux also has advanced editors like Vim, but Nano is simpler.  
This is line 10 of the sample text.  
Linux is free to use and modify.  
A global community of developers keeps Linux growing.  
You can run Linux on old or new computers.  
Linux skills can boost your career in IT.  
This text is here to help you practice Nano shortcuts.
```

**How to Use It:** Copy this text, then paste it into Nano using `Ctrl + Shift + V` in your terminal.

#### 4.8.2 Opening Nano and Using the Sample Text

- Open Nano and create a file called `sample.txt`:  

```
nano sample.txt
```
- Copy and paste the sample text above into the editor.
- Save the file by pressing `Ctrl + O`, then press `Enter`.
- Exit Nano with `Ctrl + X`.

#### 4.8.3 Basic Editing

- Reopen `sample.txt`:  

```
nano sample.txt
```
- Scroll to the bottom (`Ctrl + V`).

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

- Add this new line:  
I added this line to practice editing.
- Save (Ctrl + O, Enter) and exit (Ctrl + X).

### 4.8.4 Shortcut Exercises

Use the sample text in `sample.txt` for these tasks.

#### Exercise 1: Cut and Paste a Line

- Open `sample.txt`.
- Move to line 3 using the arrow keys.
- Press Ctrl + K to cut the line.
- Move to line 6.
- Press Ctrl + U to paste the line.
- Save the changes.

#### Exercise 2: Copy and Paste Text

- Go to any line (e.g., line 5).
- Press Ctrl + 6 to start selecting text.
- Use arrow keys to highlight a few words (e.g., “valuable skill”).
- Press Alt + 6 to copy the selection.
- Move to another line (e.g., line 8).
- Press Ctrl + U to paste it.
- Save the changes.

#### Exercise 3: Jump to a Specific Line

- Press Ctrl + \_ (underscore).
- Type 10 and press Enter to go to line 10.
- Change the line to:  
This is line 10, and I got here with a shortcut!
- Save the change.

#### Exercise 4: Search for a Word

- Press Ctrl + W.
- Type “Linux” and press Enter to find the first occurrence.
- Press Ctrl + W again to jump to the next “Linux”.
- Try this a few times to explore the file.

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### Exercise 5: Navigate the File

- Press `Ctrl + A` to jump to the start of a line.
- Press `Ctrl + E` to jump to the end of a line.
- Press `Ctrl + Y` to scroll up one page.
- Press `Ctrl + V` to scroll down one page.

### 4.8.5 Nano Assessment

To test your skills:

- Create a new file called `practice.txt`:

```
nano practice.txt
```

- Paste the sample text into it.
- Use two shortcuts (e.g., cut/paste a line with `Ctrl + K` and `Ctrl + U`, and search for “Linux” with `Ctrl + W`).
- Go to line 15 and add:

```
I'm getting the hang of Nano now!
```

- Save and exit.



# SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

## 5 Managing packages/software in Ubuntu

### 5.1 Upgrading existing packages

- Updating package/software status  
`sudo apt update`
- List down upgradable package/software  
`apt list --upgradable`
- Upgrade those packages/software  
`sudo apt upgrade`

### 5.2 Installing software package

We also can use the same command, which is **apt** to install new application either from repositories or from downloaded files. The command is:

```
sudo apt install <package/software name>
```

The **build-essential** package in Ubuntu is a metapackage that contains essential tools and libraries required for compiling and building software from source code. Installing build-essential ensures that you have the necessary tools for basic software development and for compiling software on your system.

The package includes the following components:

- **gcc**: The GNU Compiler Collection, which contains compilers for C, C++, Objective-C, Fortran, Ada, and others.
- **g++**: The GNU C++ compiler, a part of the GNU Compiler Collection.
- **make**: A utility that automates the process of building executable programs and libraries from source code.
- **libc6-dev**: The C standard library, which provides essential functions for C programs.
- **dpkg-dev**: Development tools for creating, building, and managing Debian packages.

This build-essential metapackage is prerequisite for next exercise, which is writing, compiling and running C and C++ programs.

To install build-essential metapackage, run this command:

```
sudo apt install build-essential
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

# 6 Writing and Compiling Simple C/C++ Code

## 6.1 Hello World Example in C

1. Create a new directory called **workspace** in your home directory (e.g :- /home/user/)

```
mkdir workspace
```

2. Go inside new workspace directory

```
cd workspace
```

3. Create a new file called hello.c and edit using Nano text editor

```
nano hello_world.c
```

4. In the Nano text editor, type the following C code:

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

5. Save the file and exit Nano

To save your C program, press `^O` (Ctrl + O). Nano will prompt you to confirm the file name, which should be `hello_world.c`. Press `Enter` to save the file. Next, exit Nano by pressing `^X` (Ctrl + X).

6. Compile the C program

Now that your C program is saved and GCC is installed, compile the program using the following command:

```
gcc hello_world.c -o hello_world
```

7. Run the compiled program

Execute the compiled program by entering the following command:

```
./hello_world
```

Your terminal should display the output:

```
Hello, World!
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 6.2 Hello World Example in C++

1. Make sure you are inside workspace directory. Use **pwd** command to check the current location. If not, just using **cd** command to go inside directory.

```
cd workspace
```

2. Create a new file called `hello.c` and edit using nano editor

```
nano hello_world.cpp
```

3. In the Nano text editor, type the following C code:

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello World!" << endl;
}
```

4. Save the file and exit Nano

To save your C++ program, press `^o` (Ctrl + O). Nano will prompt you to confirm the file name, which should be `hello_world.cpp`. Press `Enter` to save the file. Next, exit Nano by pressing `^x` (Ctrl + X).

5. Compile the C++ program

Now that your C++ program is saved and GCC is installed, compile the program using the following command:

```
g++ hello_world.cpp -o hello_world2
```

6. Run the compiled program

Execute the compiled program by entering the following command:

```
./hello_world2
```

Your terminal should display the output:

```
Hello, World!
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 6.3 Getting Input from Argument in C

1. Make sure you are inside workspace directory. Use **pwd** command to check the current location. If not, just using **cd** command to go inside directory.

```
cd workspace
```

2. Create a new file called `argument.c` and edit using Nano text editor

```
nano argument.c
```

3. In the Nano text editor, type the following C code:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    // Check if there are enough arguments (name and age)
    if (argc < 3) {
        printf("Usage: %s <name> <age>\n", argv[0]);
        return 1;
    }

    // Extract name and age from arguments
    char *name = argv[1];
    int age = atoi(argv[2]); // Convert age string to integer

    // Display the information
    printf("Hello, %s! You are %d years old.\n", name, age);

    return 0;
}
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 4. Save the file and exit Nano

To save your C program, press `^O` (Ctrl + O). Nano will prompt you to confirm the file name, which should be `argument.c`. Press `Enter` to save the file. Next, exit Nano by pressing `^X` (Ctrl + X).

### 5. Compile the C program

Now that your C program is saved and GCC is installed, compile the program using the following command:

```
gcc argument.c -o argument
```

### 6. Run the compiled program

Execute the compiled program by entering the following command (the programming name, following with name and age as argument.):

```
./argument farkhana 23
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 6.4 Getting Input from Keyboard in C

1. Make sure you are inside workspace directory. Use **pwd** command to check the current location. If not, just using **cd** command to go inside directory.

```
cd workspace
```

2. Create a new file called `keyboard.c` and edit using Nano text editor

```
nano input_keyboard.c
```

3. In the Nano text editor, type the following C code:

```
#include <stdio.h>

int main() {
    char name[50]; // Buffer to store the name
    int age;

    // Prompt for name
    printf("Enter your name: ");
    fgets(name, 50, stdin); // Read name from keyboard

    // Remove any trailing newline from the name
    name[strcspn(name, "\n")] = '\0';

    // Prompt for age
    printf("Enter your age: ");
    scanf("%d", &age); // Read age from keyboard

    // Display the information
    printf("Hello, %s! You are %d years old.\n", name, age);
    return 0;
}
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 7. Save the file and exit Nano

To save your C program, press `^O` (Ctrl + O). Nano will prompt you to confirm the file name, which should be `keyboard.c`. Press `Enter` to save the file. Next, exit Nano by pressing `^X` (Ctrl + X).

### 8. Compile the C program

Now that your C program is saved and GCC is installed, compile the program using the following command:

```
gcc keyboard.c -o keyboard
```

### 9. Run the compiled program

Execute the compiled program by entering the following command:

```
./keyboard
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 6.5 Simple Arithmetic Calculation in C

1. Make sure you are inside workspace directory. Use **pwd** command to check the current location. If not, just using **cd** command to go inside directory.

```
cd workspace
```

2. Create a new file called `multiply.c` and edit using Nano text editor

```
nano multiply.c
```

3. In the Nano text editor, type the following C code:

```
#include <stdio.h>
#include <stdlib.h> // For error handling

int main(int argc, char *argv[]) {
    // Only accept three arguments (program name, two numbers)
    if (argc != 3) {
        printf("Usage: %s <number1> <number2>\n", argv[0]);
        return 1; // Exit with an error code
    }

    // Convert the arguments to integers
    int num1 = atoi(argv[1]);
    int num2 = atoi(argv[2]);

    // Perform the multiplication
    int result = num1 * num2;

    // Display the result
    printf("%d * %d = %d\n", num1, num2, result);

    return 0;
}
```



## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 4. Save the file and exit Nano

To save your C program, press `^O` (Ctrl + O). Nano will prompt you to confirm the file name, which should be `multiply.c`. Press `Enter` to save the file. Next, exit Nano by pressing `^X` (Ctrl + X).

### 5. Compile the C program

Now that your C program is saved and GCC is installed, compile the program using the following command:

```
gcc multiply.c -o multiply
```

### 6. Run the compiled program

Execute the compiled program by entering the following command (program name and 2 numbers as input. In this example, the input is number 3 and 5):

```
./multiply 3 5
```

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 6.6 Linux Kernel Info in C

1. Make sure you are inside workspace directory. Use **pwd** command to check the current location. If not, just using **cd** command to go inside directory.

```
cd workspace
```

2. Create a new file called `hello.c` and edit using nano editor

```
nano kernel_info.c
```

3. In the Nano text editor, type the following C code:

```
#include <stdio.h>
#include <sys/utsname.h>
#include <stdlib.h>

int main() {
    struct utsname kernel_info;

    // Get kernel information
    if (uname(&kernel_info) == -1) {
        perror("uname");
        exit(EXIT_FAILURE);
    }

    // Display the kernel information
    printf("System name: %s\n", kernel_info.sysname);
    printf("Node name: %s\n", kernel_info.nodename);
    printf("Kernel release: %s\n", kernel_info.release);
    printf("Kernel version: %s\n", kernel_info.version);
    printf("Machine hardware: %s\n", kernel_info.machine);

    return 0;
}
```

4. Save the file and exit Nano

To save your C program, press `^o` (Ctrl + O). Nano will prompt you to confirm the file name, which should be `kernel_info.c`. Press Enter to save the file. Next, exit Nano by pressing `^x` (Ctrl + X).

## SECR2043 - OPERATING SYSTEMS (Lab Exercise)

Written by: Dr. Farkhana Muchtar

### 5. Compile the C program

Now that your C program is saved and GCC is installed, compile the program using the following command:

```
gcc kernel_info.c -o kernel_info
```

### 6. Run the compiled program

Execute the compiled program by entering the following command:

```
./kernel_info
```

Your terminal should display the output:

```
System name: Linux
Node name: os
Kernel release: 5.15.0-67-generic
Kernel version: #74-Ubuntu SMP Wed Feb 22 14:14:39 UTC 2023
Machine hardware: x86_64
```