

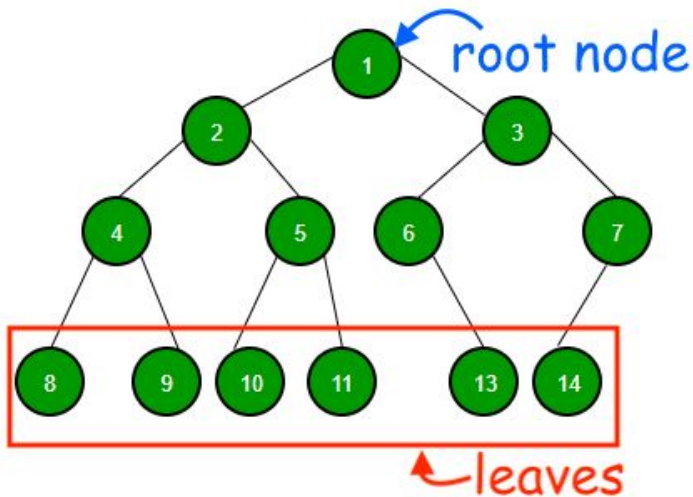


Command Line Interface Basics

Lab 01 - CS F111 (Computer Programming)

Preliminary: **Directory trees**

Let's explore a new, more formal way of thinking about how your computer stores files in folders (directories). To visualise this, we use a special kind of graph called a **tree**.



A **tree** is made up of labelled points called **nodes** or **vertices** (shown in green in the figure at the left) and lines connecting some of them, called **edges**.

A tree has a single uppermost vertex called its **root node**.

The root node is connected to one or more nodes in the level immediately below it, called its **children**. Each of these children may have its own children. These children may also have their own children, and so on...!

Nodes without children are called leaf nodes or **leaves**.

Image courtesy: <https://www.geeksforgeeks.org/binary-tree-data-structure/>

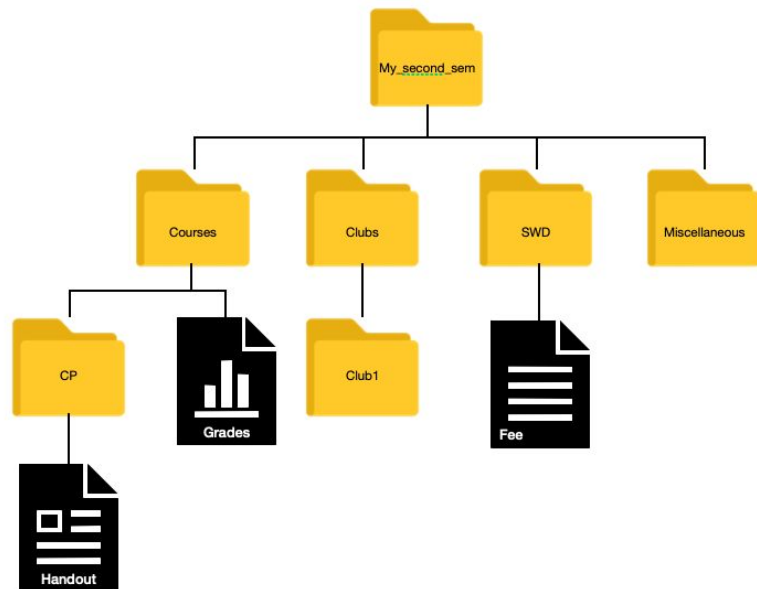
Preliminary: **Directory trees**

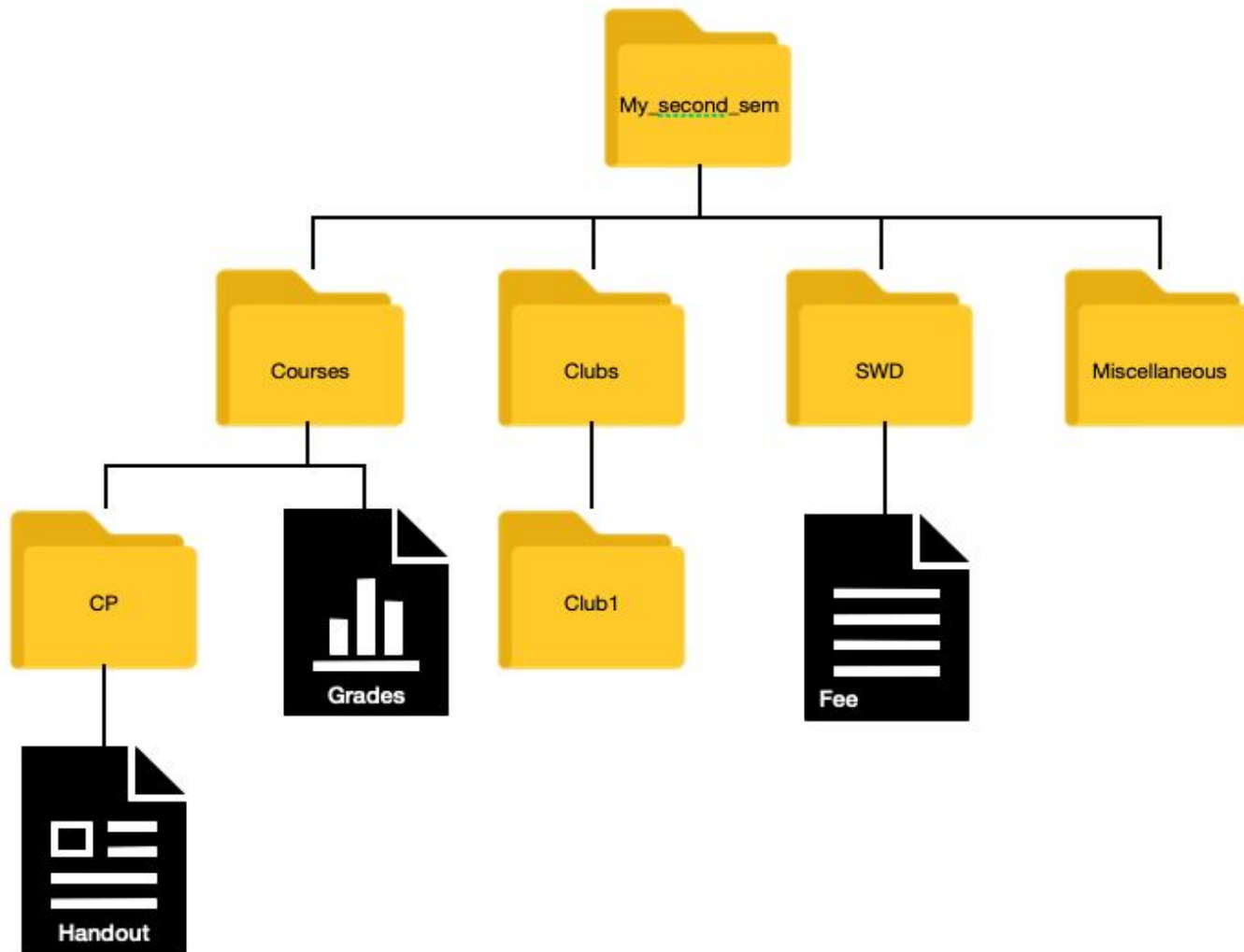
Now imagine a tree where every node represents a directory (folder) or file on your disk, and every child of a directory is contained in it.

This kind of a tree is called a **directory tree**.

The root node of this tree is called the **root directory**.

What do you observe?

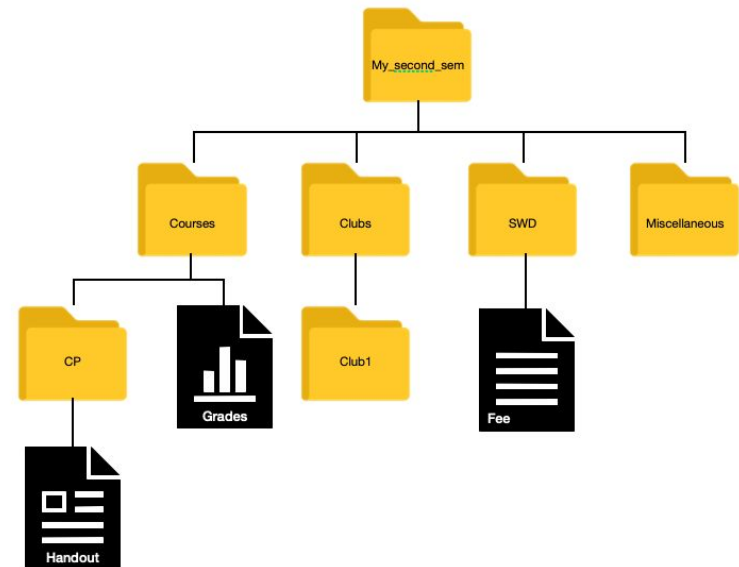




What do you observe?

Preliminary: **Directory trees**

- Every directory or file except the root directory has a unique parent
- But not every directory has a unique child
- Every file is a leaf node
- But not every leaf node is a file!
- Most importantly, for every file or directory **there is a unique path** starting from the root directory and ending at it.



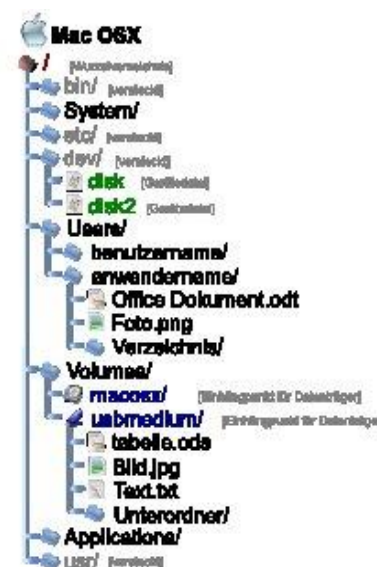
Preliminary: **Filepaths**

This unique path starting at the root node and ending at the file/directory is called its **absolute filepath**.

The root directory is common for all files on your disk, and its name depends on the operating system.

The absolute filepath is then expressed in the form

root/directory1/directory2/.../filename



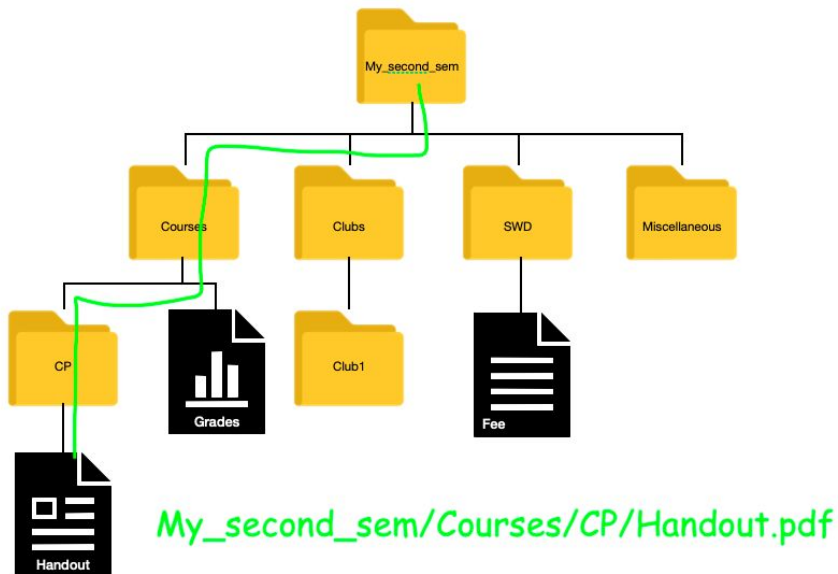
You might have come across these on your computer in the form `C:\Users\Person1\ilovecp.txt`

The root directory for Windows is in the form `drive:\` but for Linux it is simply `/`

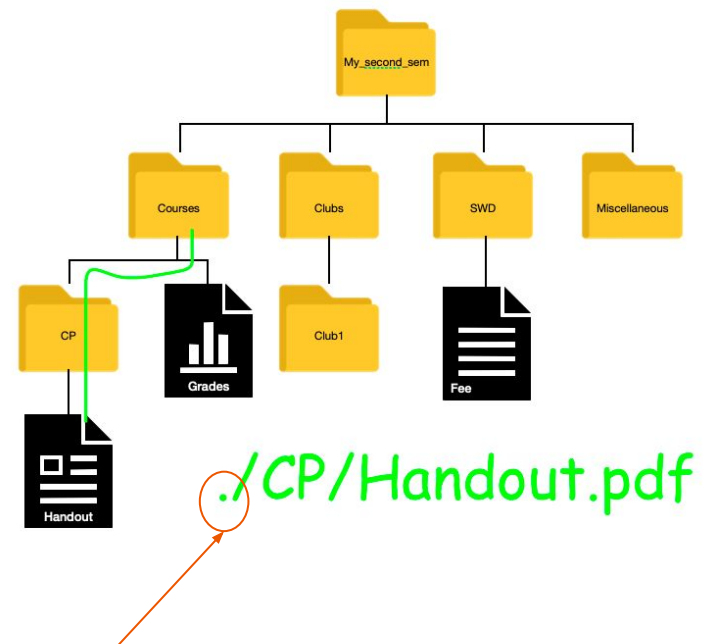
Preliminary: Filepaths

But sometimes when you are already in a directory, you might want the path from the current directory to the file you are looking for. This is called the **relative filepath** of the file to the current directory.

Absolute filepath of Handout.pdf



Relative filepath of Handout.pdf to Courses



Why the period?



Preliminary: **..** and **~**

In a filepath, a single period (.) refers to the **current directory**. For example, when you are in `Courses`, `.` refers to `Courses`, `./CP` refers to its child `CP`, `./CP/Handout.pdf` refers to `CP`'s child `Handout.pdf`, and so on.

A double period (..) refers to the **parent directory** of the current directory. For example, when your current directory is `Courses`, `..` refers to `My_second_sem`, `../Clubs` refers to `My_second_sem/Clubs`, and so on.

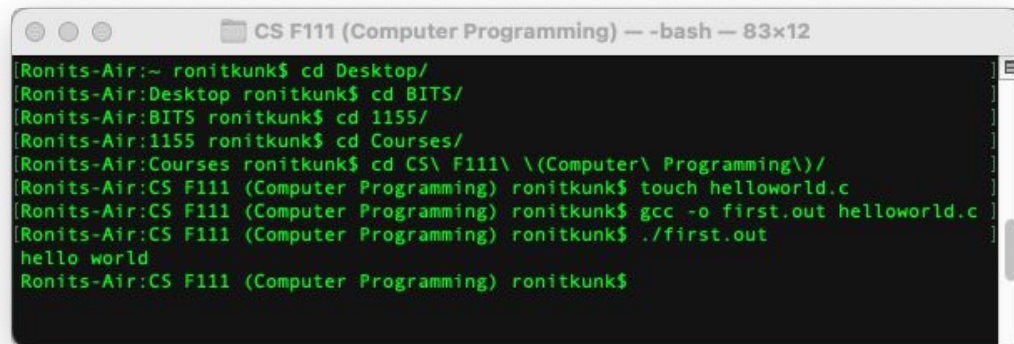
A tilde (~) refers to the user's home directory.

In Linux, the root directory is given by `/`

Terminal, Shell and Kernel

A **terminal application** (or console) is a programme you use to interact with your computer using a text-based interface.

This is done using text-based inputs called **command lines** that achieve some specific task.



```
[Ronits-Air:~ ronitkunk$ cd Desktop/
[Ronits-Air:Desktop ronitkunk$ cd BITS/
[Ronits-Air:BITS ronitkunk$ cd 1155/
[Ronits-Air:1155 ronitkunk$ cd Courses/
[Ronits-Air:Courses ronitkunk$ cd CS\ F111\ \ (Computer\ Programming\)/
[Ronits-Air:CS F111 (Computer Programming) ronitkunk$ touch helloworld.c
[Ronits-Air:CS F111 (Computer Programming) ronitkunk$ gcc -o first.out helloworld.c
[Ronits-Air:CS F111 (Computer Programming) ronitkunk$ ./first.out
hello world
[Ronits-Air:CS F111 (Computer Programming) ronitkunk$
```

An active terminal running on a computer



Terminal, Shell and Kernel

The terminal itself does not know what to do with the command lines you enter!

It passes them to another programme called a **shell**, which 'translates' these command lines into a form that can be understood by yet another programme called the kernel.

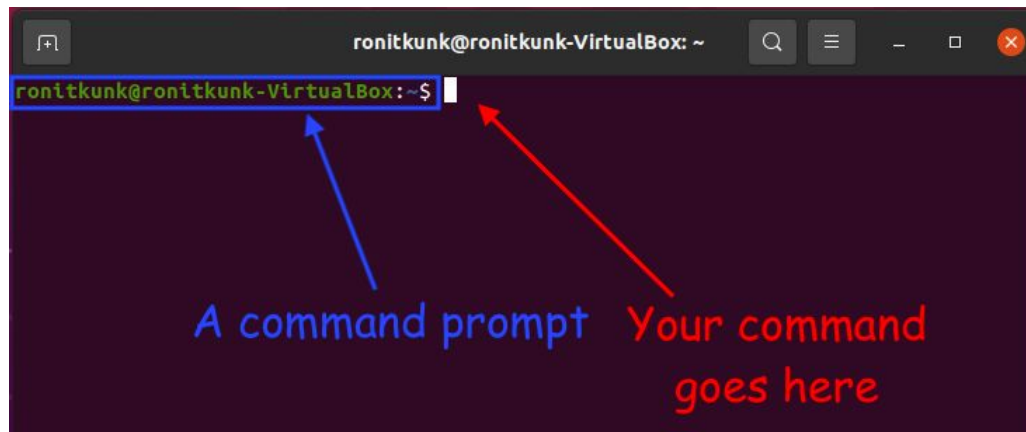
The **kernel** then works closely with your computer's hardware to perform the specific task specified by the command line.

For this session you only need to be concerned with the terminal.

The structure of a command line

When you first open a new terminal, it gives you a **command prompt**, a piece of text usually ending with a \$, > or % (depending on your shell) that indicates it is ready to accept a new command line. A prompt is also given after every subsequent command line is executed.

Recall that a tilde (~) refers to the logged in user's home directory. The tilde in the image below shows that your home directory is the current directory (working directory).

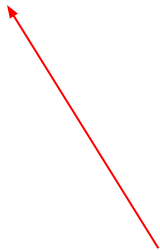




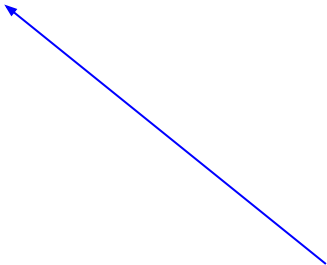
The structure of a command line

Every command line follows a standard structure.

```
$ command --flag1 --flag2 . . . -opt1 optarg -opt optarg . . . arg1 arg2 . . .
```




A command specifies the broad task you want to do.



flags and options are used to modify the behaviour of the command.

flags are preceded by a double hyphen --
options are preceded by a single hyphen - and
sometimes take **arguments** (input values) that must
be placed immediately after, separated by a space



An argument is an input taken by the command

To autocomplete the command line, use the **Tab** key.

To return to the start of the command line, use the **Home** key.



Let's get started

For this course, we'll be working in the terminal that's included in VSCode.

No need to worry - this works just like the system terminal, but it directly opens in your working folder and saves the effort of having to navigate all the way to it through a series of commands.

To start, use VSCode 'Open Folder...' menu item to open the folder where you'll be adding files.

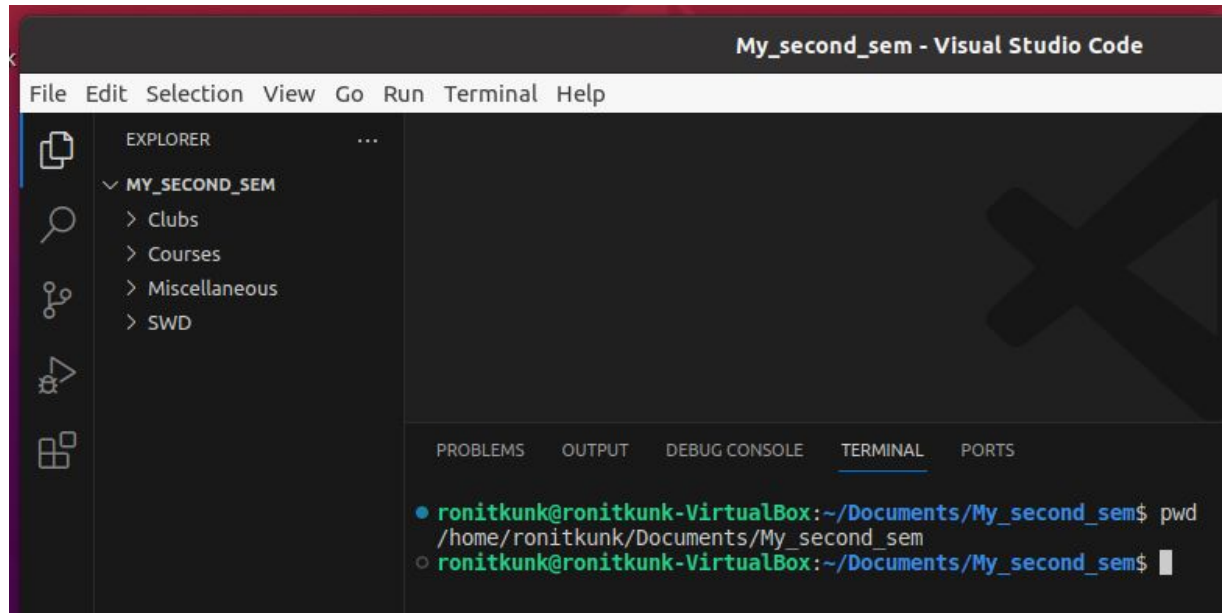
Then, go to the Menu Bar > Terminal > Create New Terminal to open a new terminal.

You will be greeted with a command prompt.

Let's get familiar with a few common terminal commands.

Commands: pwd

`pwd` (short for **p**rint **w**orking **d**irectory) prints the absolute file path to the current directory. For example, entering `pwd` immediately after opening `My_second_sem` in VSCode prints the absolute file path to `My_second_sem` on the terminal.

A screenshot of the Visual Studio Code interface. The title bar at the top reads "My_second_sem - Visual Studio Code". The menu bar includes "File", "Edit", "Selection", "View", "Go", "Run", "Terminal", and "Help". The left sidebar shows the "EXPLORER" view with a tree structure under "MY_SECOND_SEM" containing subfolders: "Clubs", "Courses", "Miscellaneous", and "SWD". The main editor area is dark and contains a large, faint "X" watermark. At the bottom, the "TERMINAL" panel is active, showing a shell prompt. The first line of the terminal shows the command `pwd` being executed, and the second line shows the output: `/home/ronitkunk/Documents/My_second_sem`. The prompt for the second line is `ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$`.

```
My_second_sem - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
  MY_SECOND_SEM
    > Clubs
    > Courses
    > Miscellaneous
    > SWD
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$ pwd
/home/ronitkunk/Documents/My_second_sem
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$
```



Commands: ls

The `ls` command lists all the items in the working directory.

```
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$ ls
Clubs  Courses  Miscellaneous  SWD
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$
```

The `ls` command can also be used to list the items in a directory other than the working directory. For this, you need to also provide the relative path of the other directory to the working directory as an argument (input) separated by a whitespace.

For example, the relative path of `Courses` to `My_second_sem` is `./Courses`. Using `ls ./Courses`, you can list the items in `Courses` while still being in `My_second_sem`!

```
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$ ls ./Courses
CP  Grades.xlsx
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$
```



Commands: cd

Sometimes, you might want to change the working directory itself. For this, you need to use the `cd` command. The argument is the relative path of the directory you want to change to. For example, here, we will change the directory from `My_second_sem` to `Clubs`.

```
● ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$ cd ./Clubs/  
○ ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Clubs$ █
```

Here the `.` refers to the current directory (`My_second_sem`).

What if you want to go 'out' of the working (current) directory? To do this, you need to go to the parent directory of the working directory. Recall that a double period `..` refers to the parent directory.

```
● ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Clubs$ cd ..  
○ ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem$ █
```


Commands: mkdir

To make a new directory inside the working directory, use the `mkdir` (make directory) command. The argument is the name of the new directory.

```
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ mkdir Thermo
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ ls
CP Grades.xlsx Thermo
```

There is a problem: When there is a space in the name of the directory, the two words separated by the space are read as separate arguments.

```
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ mkdir Electrical Sciences
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ ls
CP Electrical Grades.xlsx Sciences Thermo
```

undesirable!!

To fix this, there are some alternatives:

- Precede every space with a backslash \
- OR
- Enclose the name in double quotes

```
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ mkdir Electrical\ Sciences
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ ls
CP 'Electrical Sciences' Grades.xlsx Thermo
```



Commands: rmdir

The `rmdir` command (short for **re**move **di**rectory) is used to remove an **empty** directory from the working directory. The name of the directory to be removed is the argument.

```
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ ls
  CP  'Electrical Sciences'  Grades.xlsx  Thermo
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ rmdir Electrical\ Sciences/
• ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ ls
  CP  Grades.xlsx  Thermo
○ ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses$ █
```



Commands: touch, cat and rm

To make a new file inside the working directory, use the `touch` command. The argument is the name of the new file with its extension.

To view the file, use the `cat` command (short for concatenate) followed by the complete filename.

To remove the file, use the `rm` command (short for remove) followed by the complete filename. Be very careful while using `rm`!

```
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses/CP$ touch test.txt
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses/CP$ cat test.txt
this is the content of touch.txt!!!ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses/CP$ rm test.txt
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses/CP$ ls
Handout.pdf
```



Commands: chmod

`chmod` is a command that lets you change the permissions of a file or directory to all types of users.

Here's the syntax of the `chmod` command:

```
chmod <Operations> <File/Directory Name>
```

You can grant or revoke the permission by replacing the Operations in the above command.

These operations need to be preceded with a '+' or '-' operator.

'+' indicates adding a new permission,

'-' indicates removing an existing permission.



Permissions

- u – Grant permission to a user
- g – Grant permission to a group (A Group of users)
- o – Grant permission to others (who do not come under either of the above).
- r – Grants read permission
- w – Grant write permission
- x – Grant execute permission

Here's an example: `chmod +r sample.txt`

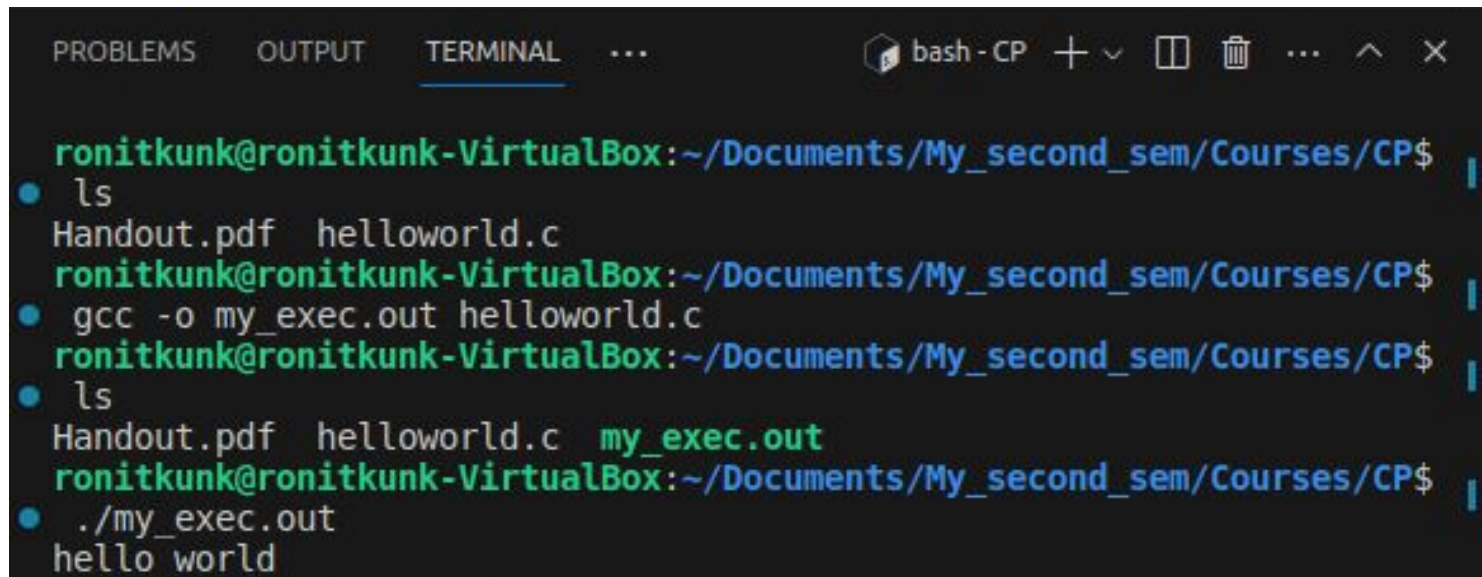
The above command adds read permission for the `sample.txt` file.

Commands: gcc

GCC, the GNU Compiler Collection, works together with a few other programs to compile the C source code file that you can understand into an executable file (Windows - .exe; Linux, macOS - .out) specific to your computer's architecture.

The name of the executable file can be chosen by adding the option -o followed by a complete name for the file.

To run an executable, use `./` followed by its name (no spaces).



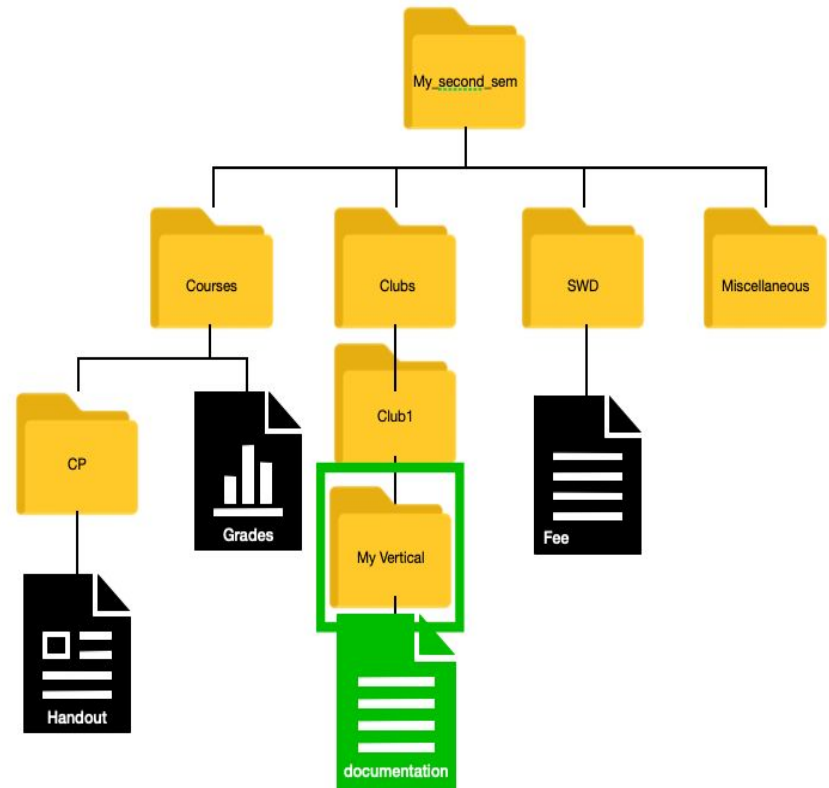
```
PROBLEMS  OUTPUT  TERMINAL  ...  bash - CP  +  v  [ ]  [X]  ...  ^  X

ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses/CP$
• ls
Handout.pdf helloworld.c
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses/CP$
• gcc -o my_exec.out helloworld.c
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses/CP$
• ls
Handout.pdf helloworld.c my_exec.out
ronitkunk@ronitkunk-VirtualBox:~/Documents/My_second_sem/Courses/CP$
• ./my_exec.out
hello world
```

Practice exercise 🧐

Without changing your working directory from CP:

1. create a new directory 'My Vertical' in Clubs
2. create a file in My Vertical called 'documentation.txt'



```
1. mkdir ../../Clubs/Club1/My Vertical
OR
mkdir ../../Clubs/Club1/"My Vertical"
2. touch ../../Clubs/Club1/My Vertical/documentation.txt
```