DISCUSSION SECTION

WEEK 1

COMMAND LINE INTERFACE (CLI)

Command Line Interface…

- is a text-based interface where you can input commands that interact with a computer's operating system (OS).

- is different on various operating systems

    * MacOS: Mac Terminal

    * Windows: Windows PowerShell or Command Prompt

    * Linux: Linux Bash Shell

    * Microsoft Azure: Azure CLI Bash

A few CLI applications…

1. Configuration of your IP address
2. Send and receive emails
3. Package Management

   * to install, update and remove software packages

4. Text Processing

   * to search within files (like Command + F)

The Standard

- Majority of developers & companies use Linux or MacOS. So, get used to Bash!
- Linux/MacOS is more comfortable; programming tools are easier to use with both e.g., Node.js with NVM
- Commands are much simpler to write in a Linux/MacOS terminal as supposed to, for example, a Windows PowerShell

`wc -l filename` VS. `(Get-Content filename | Measure-Object -Line).Lines`

Some Background about Linux File Systems

- The Linux file system can be viewed as a **tree** like structure.

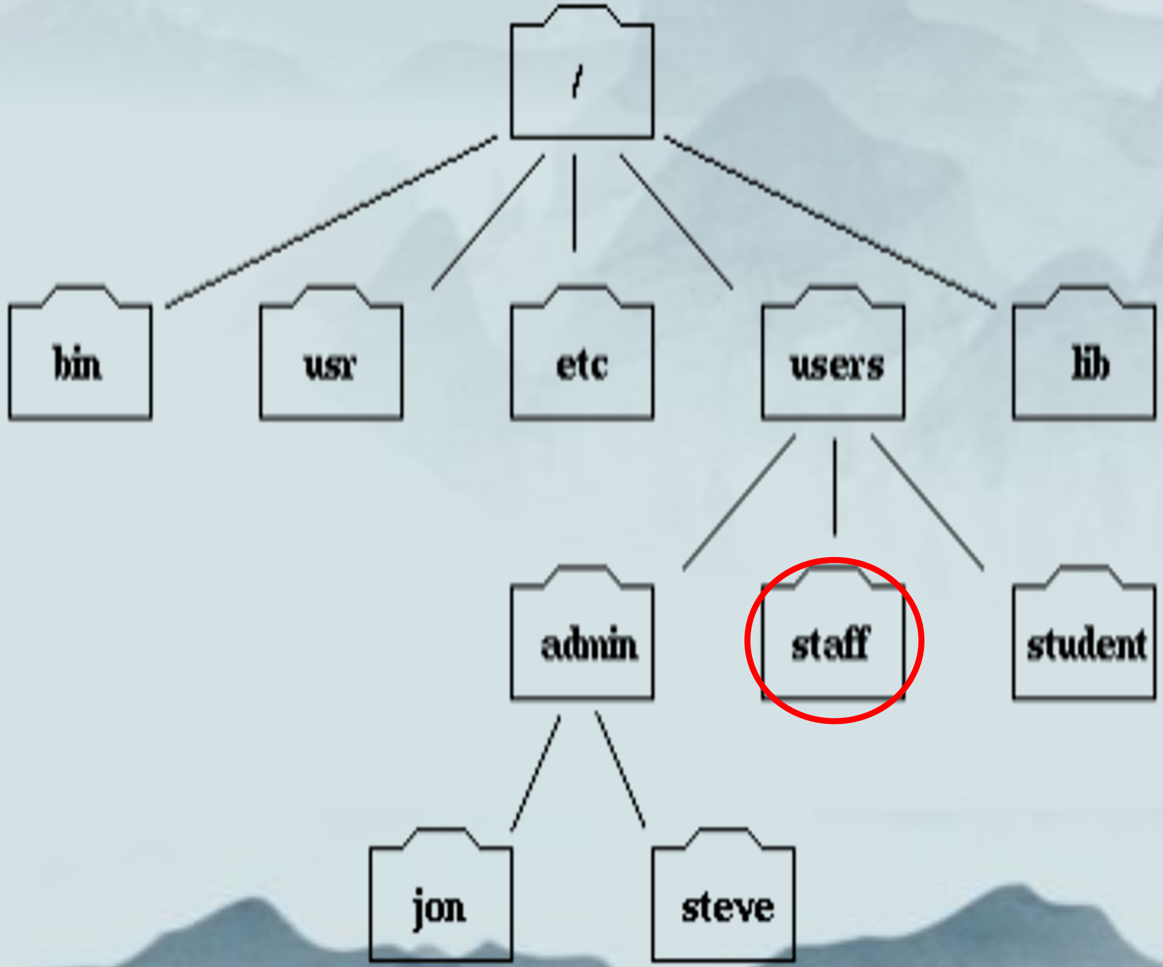
- The system is made of directories (folders), subdirectories and files.

- **~** is the *home directory*. For the purpose of this class, all work will be done in the path **~/**

File System Overview

Current working directory

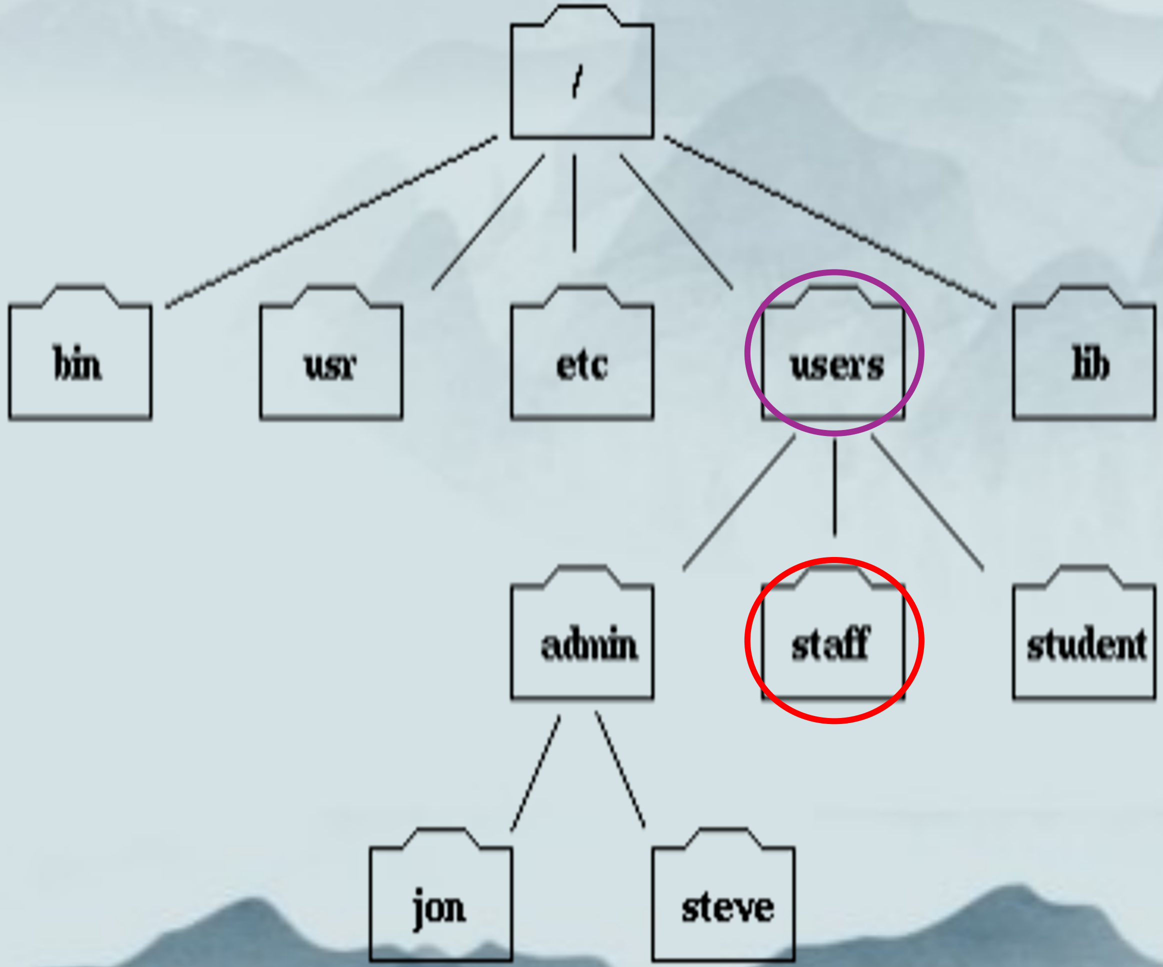
Denoted by “.”



File System Overview

"Parent directory"

Denoted by ".."



A few Shell Commands…

- pwd*
- cd*
- man
- ls*
- mkdir*
- rm*
- rmdir*
- mv*
- touch*
- cp
- cat
- echo*
- grep

*Frequently used in CSC 211

pwd

Prints the current working directory

cd

Change Directory – change the current working directory to a specific Folder

mkdir

Create new folder(s), if they do not already exist

touch

Create new file(s)

rm

Remove files (delete/unlink)

rmdir

Remove/delete folder; this command will only work if the folder is empty

rm –r folderName to delete non-empty folders. **Warning: this is irreversible!**

mv

Move or rename files or directories

echo

Display message on screen, writes each given String to standard output, with a space between each, and a newline after the last one

clear

Clear the entire terminal window

open

Used to launch files, folders (multiple too), URLs, applications, and others..

cat

Concatenate and print (display) the content of files

nano

Opens a file in the nano text editor. If the file does not exist, nano will create it for you.

Exercise 1 (10 minutes)

Provide a sequence of commands to:

i. Navigate to your Desktop directory
ii. Create a folder named **Exercise-1** and navigate to that folder
iii. Create a file named **bashIntro.txt** and add the text “I am learning bash!” to the file (*Hint: use nano!)*
iv. Make two copies of **bashIntro.txt,** named **copy1.txt** and **copy2.txt**
v. Delete **copy2.txt**; Rename **copy1.txt** to **bashCopy.txt**
vi. Display the contents of **bashCopy.txt**
vii. Delete the **Exercise-1** folder

Command Line Operators

Shell commands are cool, right? There's a lot you can do with just the list in the section above! But what if I told you it gets even better?

Similar to logical operators in C/C++, or any language for that matter, the command line supports command line operators for more efficient or even multiple operations. Here are a few of the most frequented command line operators...

Operator	Information	Example
|	The **pipe** operator directs the output of the preceding command as input to the succeeding command. It is most commonly used to filter data with the grep command.	`cat test | grep -i "makeuseof"`
&&	This operator functions in similar ways to the semicolon operator except, unlike the semicolon operator, AND operator will execute commands only if the preceding command was successfully executed.	`pwd && mkdir test && cd test && bad_command && ls`
||	The OR operator will execute the command that follows only if the preceding command fails, i.e., returns an exit code of 0. It functions like a logical OR gate, which returns a value of 1 when the input is 0.	`bad_command || ls`
>>	The redirection operators redirect output or input to a file either by re-writing the file or by appending to it. If you want to re-write a file, then you have to use the single angle bracket (>) syntax. If you want to append to a file, you'll have to use the double angle bracket syntax (>>).	`echo "dsd" > test ; echo "bssss" >> test`

Exercise 2 (10 minutes)

DO IT ALL AT ONCE WITH COMMAND LINE OPERATORS!

i. Navigate to your Desktop directory
ii. Create a folder named **Exercise-1** and navigate to that folder
iii. Create a file named **bashIntro.txt** and add the text "I am learning bash\!" to the file (*Hint: use redirection operator!)*
iv. Make two copies of **bashIntro.txt,** named **copy1.txt** and **copy2.txt**
v. Delete **copy2.txt**; Rename **copy1.txt** to **bashCopy.txt**
vi. Display the contents of **bashCopy.txt**
vii. Delete the **Exercise-1** folder

Now you should be able to compile and run your C++ code at once!

```
g++ main.cpp -o main && ./main
```