Notes 1

What is Markdown?

Markdown is a markup language that lets us write plain text documents with a few lightweight formatting options createed in 2004 by John Gruber.

What is Git?

Git is the most widely used modern distributed version control system in the world today. It is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel.

What is GitHub?

GitHub is the cloud-based version control system for hosting Git repositories for collaborations with other developers to store, share, and work together to write code.

What is Slack?

Slack is a instant messaging program designed by Slack Technologies for team communication that allows users to chat, share files, and integrate with other software and owned by Salesforce.

Notes 2

1. What is an Operating System?

An **operating system** provides all fundamental software features of a computer. It enables us to use the computer's hardware providing us the basic tools that make the computer useful. All those features rely on the OS's kernel and other OS features are owed to additional programs that run atop the kernel.

2. What is a kernel?

An **OS kernel** is a software component that's responsible for managing low-level features of the computer, including the following managing system hardware, memory allocation, CPU time, and program to program interaction.

3. Which other parts aside from the kernel identify an OS?

- **Command-Line Shells**
  - This was the de facto way of using computers before the Graphical Interface was invented. CMDs work by typing commands in a shell. In Linux, the entire system can be control via the CLI.
- **Graphical User Interfaces**
  - GUIs rely on icons, menus, and a mouse pointer for the user interaction. Linux relies on a GUI known as the X Window System in combination with desktop environments program suites.

- **Utility and Productivity Programs**
  - Tools like web browsers, document processors and text editors.
- **Libraries**
  - Libraries are collections of programming functions that can be used by a variety of programs.

4. What is linux and linux distribution?

Linux is a **Unix-like Operating System** popular in academic and business environments which consists of a kernel, libraries, and utilities that make up the entire operating system.

Linux distribution is any operating system that runs the Linux kernel or a complete Linux system package. Some popular linux distributions include **Arch, CentOS, Debian, Fedora, openSUSE, Red Hat, Slackware, Ubuntu** and many more.

5. List at least 4 linux characteristics:

- Linux is an **open source software** available **free of charge**.
- Linux is **highly scalable** and customizable.
- Linux includes many of the **Unix tools** including many important Internet server programs and programming languages out of the box.
- Linux can be **installed on almost any system** as it supports almost every processor architecture.

6. What is Debian?

Debian is an all-volunteer organization dedicated to developing **free software** and promoting the ideals of the Free Software community.

7. List and define the different types of licensing agreements

- **Open Source**: The source code is distributed with the software and the software maybe distributed for a fee or free.
- **Closed Source**: The source code is not distributed with the software and the user is restricted from modifying the code.
  - **Freeware**: The software that is free but the source code is not available.
  - **Shareware**: The software that is free on a trial basis.

8. What is Free Software? Define the 4 freedoms.

The software is distributed with the source code and the software can be free of charge or obtained by a fee.

The **Free Software Foundation** (FSF) defines four software freedoms:

- **Freedom 0**: use the software for any purpose
- **Freedom 1**: examine the source code and modify it as you see fit
- **Freedom 2**: redistribute the software
- **Freedom 3**: redistribute your modified software

9. What is virtualization?

Virtualization is defined as **Cloud computing** or creating virtual versions of something. It lets **multiple OSs run on one physical machine** at the same time. It also allows administrators to **divide the hardware** and create multiple computers **inside a single physical computer**.

Notes 3

What is a graphical user interface (GUI)?

A **graphical user interface** (GUI) is a set of programs that allow a user to interact with the computer system via icons, windows, and various other visual elements.

What is a desktop environment?

A **desktop environment** is an implementation of the desktop metaphor made of a bundle of programs running on top of a computer operating system, which shares a common GUI, sometimes described as a graphical shell.

What is the command line interface (CLI)?

The **command line interface** is a text-based interface where the user interacts with the computer by inputting commands instead of using mouse or graphical icons.

How do I access the command line interface (CLI)?

There are two ways to access the CLI:

- **Terminal Emulator:** A **terminal emulator** is a software program that allows us to access the Linux Command Line Interface (CLI) when using the Graphical User Interface (GUI).
- **Linux Console:** The **linux console** is a text based interface that does not require the desktop or graphics at all.

What is a virtual console?

A **virtual console** is a terminal session that runs in Linux system memory.

What is a terminal emulator?

A **terminal emulator** is a software program that allows us to access the Linux Command Line Interface (CLI) when using the Graphical User Interface (GUI).

Some terminal emulators are:

- GNOME
- Konsole
- Terminology
- RXVT-Unicode
- TILIX
- Kitty

What is bash?

The **bash shell** is a program that provides interactive access to the Linux system which runs as a regular program and is started whenever a user logs in.

What is the shell prompt?

A **shell prompt** is the text that appears in a command-line interface(CLI) indicating that the system is ready to accept a command from user. It includes the **username@machinename**, followed by the current working directory and a dollar sign.

Basic Commands:

clear

- **Definition**: clears the terminal screen
- **Usage**: `clear`
- **Examples**:
  - To clear the screen:
    - `clear`

---

echo

- **Definition**: Displays or prints text on the screen
- **Usage**: `echo` + `options` + `string to display`
- **Examples**:
  - To display **a line** of text:
    - `echo "Hello!"`
  - To display **two lines** of text:
    - `echo -e "Hello\nWorld"`
  - To display a line of text **without the new line**:
    - `echo -n "Hello There!"`

---

date

- **Definition**: Prints current date and time
- **Usage**: `date`+ `options`
- **Examples**:
  - To display current date
    - `date`
  - To display current time in rfc 5322 format
    - `date -R`

---

free

- **Definition**: Displays amount of free and used memory in the system

- **Usage**: `free` + `options`
- **Examples**:
  - To display memory utilization
    - `free`
  - To display memory utilization in **human readable** format
    - `free -h`

---

uname

- **Definition**: Prints system information
- **Usage**: `uname` + `options`
- **Examples**:
  - To print **all** information
    - `uname -a`
  - To print **kernel** information
    - `uname -s`
  - To print **hostname**
    - `uname -n`

---

history

- **Definition**: Shows command line history
- **Usage**: `history` + `options`
- **Examples**:
  - To display session history
    - `history`
  - To clear session history
    - `history -c`

---

man

- **Definition**: Shows or displays the manual page for a given command
- **Usage**: `man` + `options` + `command`
- **Examples**:
  - To open the man page of echo command
    - `man echo`
  - To open a specific man page
    - `man 5 passwd`
  - To show all available man page
    - `man -f passwd`

---

tldr

- **Definition**: Shows simplified and community-contributed examples for commands

- **Usage**: `tldr` + `options` + `command`
- **Examples**:
    - To update the tldr local cache
        - `tldr -u`
    - To view the simplified documentation of echo command
        - `tldr echo`

---

cheat

- **Definition**: Shows cheat sheets for commands
- **Usage**: `cheat` + `options` + `command`
- **Examples**:
    - To view cheat sheets for the git command
        - `cheat git`
    - To view the cheat sheets of find command in **colorize** mode
        - `cheat --colorize find`

---

hostname

- **Definition**: Shows or set the system's hostname or computer name
- **Usage**: `hostname` + `options`
- **Examples**:
    - To view current host name
        - `hostname`
    - To display **all addresses** for the host
        - `hostname -I`

---

df

- **Definition**: Displays the amount of disk space available/used on entire file systems/partitions
- **Usage**: `df` + `options`
- **Examples**:
    - To show disk usage
        - `df`
    - To display the amount of disk space in **human readable** format
        - `df -h`

---

du

- **Definition**: Displays the amount of space used by specific files or directories
- **Usage**: `du` + `options` + `file/directory`
- **Examples**:
    - To display or **summarize** the size of current directory in **human-readable** form
        - `du -sh`

- To display the size of a specific file in **human-readable** form
  - `du -h wr3.png`

---

figlet

- **Definition**: Displays text in large ASCII letters
- **Usage**: `figlet` + `text`
- **Examples**:
  - To display or create ASCII letters
    - `figlet Hello!`

Notes 4

How to install and remove software using the APT command.

APT(Advanced Package Tool) is a set of tools used for managing Debian packages such as **installing, updating, and removing the software** through command-line interface.

Formula: `sudo + apt + install/update/remove + package_name`

To Install Software:

1. Before installing any software, **first update** all the packages in the system. Use any of these two commands to update Debian.
   - `sudo apt update; sudo apt upgrade -y`
2. Then **install a package** using the following command to install any package we want.
   - `sudo apt install package_name`
   - **For example:** `sudo apt install tilix; sudo apt install bastet`

To Remove Software:

1. Use the following command to **remove a package without deleting the configuration files**.
   - `sudo apt remove package_name`
   - **For example:** `sudo apt remove cheat; sudo apt remove firefox`
2. To **completely remove a package including the configuration files** use the following command.
   - `sudo apt purge package_name`
   - **For example:** `sudo apt purge firefox`
3. After uninstalling software, **remove all unused dependencies** and **clean up** the system.
   - `sudo apt clean; sudo apt autoclean; sudo apt autoremove`

We can also **install or remove** multiple programs **at the same time** by adding the package name with space between two package and using **'+' and '-' sign** at the end of each package.

- **For example:** `sudo apt install tilix+ bastet+ cheat- firefox-`

How to create a shell script step by step including screenshots and how to run it.

Shell script is a text file containing a series of commands created so that the Linux shell can automate the execution of multiple commands.

The step by step process is as follows:

Step 1: Create the file

Open a text editor and create a script file. Then, save the file as **file_name.sh**. **For example:** `note4.sh`

Step 2: Add shell declaration

The first line in the file must be the **shell interpreter** or **shebang** to tell the system which shell to use. For **bash shell** it would be: `#!/bin/bash`

Step 3: Add your code

Write the commands we want the shell to execute when the file is run. The **formula or syntax** to write a command is : `echo + option + "string"`

For example: `#!/bin/bash echo "Hello There!" echo -e "Hey!\tHow are you?\nI am good, and yourself?" echo -e -n "Doing just fine."`

Then, save and exit.

Step 4: Run the script

To run the script **open terminal** and use the following command:
`bash~/path/to/script/script_name.sh`

For example: `bash~/Scripts/note4.sh`

Then we can see the output displayed in our terminal.

Note 5

Commands to navigate the linux file system

LS

- **Usage:**
  - `ls` is used for listing files and directories.
  - By default it will list the current directory when no directory is specified.
  - Listing means to see what is inside a directory.
- **Formula:**
  - `ls + option + directory(ies) to list`
- **Examples:**
  - See all the options of the ls command (extracted from the man page):
    - `ls --help`
  - List the current directory:
    - `ls`
  - List all the files including hidden files in current directory:
    - `ls -A`
  - Long list a directory
    - `ls -lA ~/Pictures`
  - List a directory recursively
    - `ls -R Documents/`
  - Long list a directory only
    - `ls -ld Documents/`
  - List a directory sorted by last modified
    - `ls -t Documents/`
  - List a directory sorted by file size
    - `ls -S Documents/`
  - Long list a directory excluding group and owner information, with human readable file size and sorted in reverse order.
    - `ls -lhgGr Documents/`

PWD

- **Usage:**
  - Displays the absolute path of the current working directory.
- **Formula:**
  - `pwd`
- **Examples:**
  - Print the absolute path of current working directory
    - `pwd`

CD

- **Usage:**

    - Changes the current working directory. In other words, it moves us from one directory to another.
    - By default, it will always send us to our home directory.
- **Formula:**
    - `cd + destination absolute path or relative path`
- **Examples:**
    - Go (change our current directory) to our home directory:
        - `cd` (without any arguments, cd will take us home)
        - `cd ~` (using the ~ special character. as ~ will expand to the absolute path of the user's home directory)
        - `cd $HOME` (using the $HOME environment variable)
        - `cd /home/$USER/Downloads` (using $USER environment variable in the path)
    - Go to a specified directory with absolute path:
        - `cd /usr/share/themes`
    - Go to a specified directory with relative path assuming your current working directory is /home
        - `cd aayushmas/Downloads/`
    - Go to the previous working directory. This is useful when you are working with 2 directories located far in the directory tree
        - `cd -`
    - Go to the previous directory in the directory tree. One directory above.
        - `cd ../`
    - Go to 2 directories above the directory tree
        - `cd ../../`

What is a variable?

In programming, a **variable** is a container or placeholder to store data. A variable is like a box with a label which can be used to store temporary or permanent information that we will continuously reuse in our program. **For example**, `username='aayushmas'` the variable name now stores the value `aayushmas`. Whenever the programs need to access the aayushmas's username, it can do it by referencing the variable `username`.

How do I use a variable?

- **Creating a variable:**
    - We can create or assign a variable using '**=**' sign.
    - **For example:** `name='aayushma'` (Here, the variable 'name' stores the value 'aayushma')
- **Using a variable:**
    - To use or show the value stored in a variable, add a '**$**' sign before the variable name.
    - **For example:** `echo $name` which outputs `aayushma` the value that was stored in variable 'name'.

What is an environment variable?

Environment variables store values of a user's environment and can be used in commands in the shell. These values can be unique to the user's environment which makes them ideal when writing commands that we want to use regardless of which user is using the computer.

- To see a list of our environment variables type `env`.
- To use the value stored in an environment variable we must prepend the variable name with a `$`.
- Environment variables are typed in **capital letters** to differentiate from user defined variables.
- **For example:**
  - **$USER** = stores the current's user username
  - **$HOME** = stores the absolute path of current's user home directory
  - **$PWD** = stores the absolute path of the present working directory.
  - **$OLDPWD** = stores the absolute path of the previous current working directory

What is a user defined variable?

User defined variables are created by the user and exist only in the script and subshell that runs the script. It allows us to temporarily store data and use it throughout the script-just like any other programming language.

- It can be any text string of up to 20 letters, digits, or underscore characters but they **CANNOT start with a number.**
- User variables are **case sensitive.** (eg: var1 is not Var1)
- Values are assigned using an **equal sign with no spaces.** (eg: name='Peter')
- The shell stores all values as text strings; Bash is essentially untyped.

What is the root directory?

Root directory is the first directory in the filesystem that contains the entire filesystem represented by "`/`".

What does "Parent Directory" mean?

Parent directory is a directory containing one or more directories and files.

What does "Current working directory" mean?

Current working directory is also known as the present working directory. It is the directory where we are currently working in. We are always working from a directory.

What is an absolute path? Include an example.

An **absolute path** is the location of a file starting at the root of the file system. **For example:** `/home/aayushmas/Downloads/list.txt` is the absolute path of the file `list.txt.`

What is a relative path? Include an example.

A **relative path** the location of a file starting from a child directory of the current working directory or from the current directory itself. **An example** of a relative path would be `Downloads/list.txt` assuming that the current working directory is `/home/aayushmas`

What is the difference between "Your home directory" and "The home directory"?

YOUR HOME DIRECTORY is our user's personal directory where all our files are located. Every user has it's own home directory. We have total ownership of our home directory but outside of the home directory only the root user can make changes.

- **For Example:** An absolute path, assuming that user name is aayushmas, would be `/home/aayushmas`

Whereas, **The Home Directory** is the parent directory of all the home directories. This is where all the users' home directory are.

- **For Example:** The absolute path of this directory is `/home`. Noticed that it starts at the root.

Notes 6

MKDIR

- **Usage:**
  - **mkdir** is used for creating a single directory or multiple directories (folders) **by separating each directory name with a space**.
- **Formula:**
  - `mkdir + option + the name of the directory`

  Where directory name can be:
  - Just the name of the directory if we want to create them in the **current working directory**
  - **Absolute** or **relative path** if we want to create the directory in a different location
- **Examples:**
  - Create a directory in the present working directory called wallpapers
    - `mkdir wallpapers`
  - Create a directory in a different directory using **relative path**
    - `mkdir wallpapers/ocean`
  - Create a directory in a different directory using **absolute path**
    - `mkdir ~/wallpapers/forest`
  - Create a directory with a space in the name
    - `mkdir wallpapers/new\ cars`
    - `mkdir wallpapers/'cities usa'`
  - Create a directory with a single quote in the name
    - `mkdir wallpapers/"major's mask"`
  - Create multiple directories
    - `mkdir wallpapers/cars wallpapers/cities wallpapers/forest`
  - Create a directory with a parent directory at the same time.
    - `mkdir -p wallpapers_others/movies`
  - Create a directory and display a message confirming the directory creation (**verbose output**)
    - `mkdir -pv wallpapers_others/movies`

TOUCH

- **Usage:**
  - **touch** is used for creating files an empty file or to update the timestamp of an existing file..
- **Formula:**
  - `touch + name of the file`

  Where directory name can be:
  - Just the name of the directory if we want to create them in the **current working directory**
  - **Absolute** or **relative path** if we want to create the directory in a different location
- **Examples:**
  - Create a file called list
    - `touch list`
  - Create multiple files
    - `touch list_of_cars.txt script.py names.csv`
  - Create a file using **absolute path**
    - `touch ~/Downloads/games.txt`
  - Create a file using **relative path** (assuming you pwd your home directory)
    - `touch Downloads/games2.txt`
  - Create a file with a space in its name
    - `touch "list of foods.txt"`

RM

- **Usage:**
  - **rm** is used to remove files and directories.
  - It is a powerful command that **can permanently delete files**, so it should be used with caution.
  - To remove directories use the `-r` option.
- **Formula:**
  - `rm + option + name of the file or directory to remove`

  Where file/directory name can be:
  - Just the name of the files/directories if they are located in the current working directory
  - Absolute or relative path if they are located in a different location

  Common Options:

- `-f`: Force removal without prompting for confirmation.
- `-r` or `-R`: Recursively remove directories and their contents.
- `-i`: Prompt before each removal.
- `-v`: Verbose mode, showing the files as they are removed.

- **Examples:**
  - Remove a file called list
    - `rm list`
  - Remove a file and prompt confirmation before removal
    - `rm -i list`
  - Remove all the files inside a directory and ask before removing more than 3 files
    - `rm -I Downloads/games/*`
  - Remove an empty directory
    - `rmdir Downloads/games`
  - Remove an non-empty directory
    - `rm -r Downloads/games`
  - Remove files and directories but prompt for confirmation before removing and display removal message when done
    - `rm -vir games/program.py ~/ProjectDelta`
  - Remove a **non empty directory** forcing the removal
    - `rm -rf games/`

CP

- **Usage:**
  - **cp** is used to copy files/directories from a source to a destination
  - Must use the `-r` option to copy directories

- **Formula:**
  - `cp + option + file/ directories to copy + destination`
  - `cp -r + directory to copy + destination`

  Common Options:
  - `-r` or `-R`: Recursively copy directories and their contents.
  - `-i`: Prompt before overwriting an existing file.
  - `-u`: Copy only when the source file is newer than the destination file or when the destination file is missing.
  - `-v`: Verbose mode, showing the files as they are copied.
  - `-a`: Copy files and directories, preserving attributes like timestamps and permissions.

- **Examples:**
  - To copy a file
    - `cp Downloads/wallpapers.zip Pictures/`

- To copy a directory with **absolute path**
  - `cp -r ~/Downloads/wallpapers ~/Pictures/`
- To copy the content of directory to another directory
  - `cp Downloads/wallpapers/* ~/Pictures/`
- To copy multiple files in a single command
  - `sudo cp -r script.sh program.py home.html assets/ /var/www/html/`
- To copy a directory with verbose output
  - `cp -rv ~/projectOrion/ ~/Documents/`

MV

- **Usage:**
  - **mv** is used to move and rename files and directories.
  - **mv** cannot rename more than 1 file at the time
  - **mv** can move and rename a file at the same time
  - **mv** will set the last argument as the destination or file new name
- **Formula:**
  - **Move:** `mv + option + file,directories to move + destination`
  - **Rename :** `mv + option + file/directory to rename + new name`

  Common options of the mv command:
  - `-i`: Prompt before overwriting an existing file.
  - `-u`: Move only when the source file is newer than the destination file or when the destination file is missing.
  - `-v`: Verbose mode, showing the files as they are moved or renamed.
- **Examples:**
  - To **move** a file from a directory to another using **relative path**
    - `mv Downloads/homework.pdf Documents/`
  - To **move** a directory from one directory to another using **absolute path**
    - `sudo mv ~/Downloads/theme /usr/share/themes`
  - To **move** a file from one directory to another combining **absolute path** and **relative path**
    - `mv Downloads/english_homework.docx /media/student/flashdrive/`
  - To **move** multiple directories/files to a different directory
    - `mv games/ wallpapers/ rockmusic/ /media/student/flashdrdive/`
  - To **rename** a file

        - `mv homework.docx cis106homework.docx`
    - To **rename** a file using **absolute path**
        - `mv ~/Downloads/homework.docx ~/Downloads/cis106homework.docx`
    - To **move and rename** a file in the same command
        - `mv Downloads/cis106homework.docx Documents/new_cis106homework.docx`

Notes 7

Wildcard

- **A wildcard** is a symbol used to replace or represent one or more characters in a file name.
- **There are 3 wildcards:**
    - `*` : matches zero to any number of characters
    - `?` : matches only 1 character
    - `[]` : matches 1 character from as given set

The asterisk (*) Wildcard

- The asterisk (*) matches **zero or more characters** in a filename.
- **Examples:**
    - List all the files in a given directory
        - `ls Downloads/*`
    - List all the text files in a given directory
        - `ls Downloads/*.jpg`
    - List all the text files in a given directory that start with letter f
        - `ls Downloads/w*.jpg`
    - List all the files that contain the word file in the name
        - `ls *file*`
    - Move all the files one directory to another
        - `mv ~/Downloads/Nature/* ~/Pictures/wallpapers/`
    - Copy specific files based solely on their file extension
        - `cp ~/Downloads/home/*.pdf ~/Documents/*.txt ~/Projects/school/`
    - Remove specific files
        - `rm ~/Downloads,demo*.exe ~/Videos/*music*.avi`
    - Move specific files from one directory to another
        - `mv Downloads/Movies/{*.png,*.gif} Downloads/Movies/MCU`

The ? Wildcard

- The ? wildcard meta-character matches **precisely one character.**
- The ? wildcard proves very useful when working with hidden files(dot files).
- To list all hidden files use: ls.??* which will match all files that start with a. or .. and have any character after it.
- **Examples:**
  - List all the hidden files in the current working directory
    - `ls ./.??*`
  - List all the hidden files in the parent directory
    - `ls ../.??*`
  - List all the files that have 2 characters in the file name between letters m and i
    - `ls m??i*`
  - List all the files that have a single character between letters s and a
    - `ls s?a*`
  - List all the files with a 2 letter file extension
    - `ls *.??`

The [] Wildcard

- The brackets wildcard matches a **single character in a range.**
- The brackets wildcard use the exclamation mark to reverse the match.
- For example: match everything except vowels[!aeoiu] or any character except numbers[!0-9]
- **Examples:**:
  - To match all files that have a vowel after letter m:
    - `ls m[aeiou]*`
  - To match all files that do not have a vowel after letter m:
    - `ls m[!aeiou]*`
  - To match all files that have a range of letters after l:
    - `ls l[a-z]*`
  - To match all files whose name has at least one number:
    - `ls *[0-9]*`
  - To match all the files whose name does not have a number in their file name:
    - `ls *[!0-9].*`
  - To match all files whose name begins with a letter from a-p or start with letter s or c:
    - `ls [a-psc]*`
  - To match all files whose name begins with any of these two sets of characters: letters from a-f or p-z:
    - `ls [a-fp-z]*`
  - To match all files whose name begins with any 3 combination of numbers and the current user's username:
    - `ls [0-9][0-9][0-9]$USER`

Brace Expansion

- **Brace expansion** is a feature of the bash shell that generates argument strings.
- Those strings can be used by commands to operate on files.
- They does not make calls to the operating system like wildcards do.
- They simply generate file names based on a given pattern.

How to use Brace Expansion to create entire directory structures.

- Start with an open brace
- With no spaces, type your string separating entries by command
- Close the brace
- **Examples:**
  - Create 3 different files with the same name but different file extensions
    - `touch file.{md,txt,rtf}`
  - Create 10 files in a range from 0 to 9
    - `touch file{0..9}.txt`
  - Remove specific files that start with a given keyword
    - `rm image_*{01..08}*_camera.{png,jpg}`
  - Create an entire directory tree in a single command(1 level deep)
    - `mkdir -pv project_venus/{code,source,dataset}/new`
  - Create an entire directory tree in a single command(2 level deep)
  - `mkdir -pv project_jupiter/site/{old,new}/{code/{scripts,markup},assets/{imgs,mp3,mp4}}`

Notes 8

CAT

- **Definition:**
  - The **cat command** is used for displaying the content of a file.
  - **Cat** is short for **concatenate** which is the command's intended use.
- **Usage:**
  - `cat + option + file(s) to display`
- **Examples:**
  - Display the content of a file located in ~/Documents/sample_files/
    - `cat ~/Documents/sample_files/Code/helloWorld.py`
  - Display the content of a file **with line numbers**
    - `cat -n ~/Documents/sample_files/Code/helloWorld.py`
  - Display the content of a file including **non printing characters and line endings**
    - `cat -A ~/Documents/sample_files/Code/helloWorld.py`

TAC

- **Definition:**

    - The **tac command** is used for displaying the content of a file in **reverse order**.
    - Just like cat, tac concatenates files and displays the output of the concatenation.
- **Usage:**
    - `tac + option + file(s) to display`
- **Examples:**
    - Display the content of **a file** located in ~/Documents/sample_files/ in reverse order
        - `tac ~/Documents/sample_files/Code/helloWorld.py`
    - Display the content of **multiple files** in reverse order
        - `tac ~/Documents/sample_files/Code/helloWorld.py ~/Documents/sample_files/Code/helloWorld.sh`

HEAD

- **Definition:**
    - The **head command** displays the top N number of lines of a given file.
    - By default, it prints the **first 10 lines**. If more than one file name is provided then data from each is preceded by its file name.
- **Usage:**
    - `head + option + file(s)`
- **Examples:**
    - Display the **first 10 lines** of a file
        - `head ~/Documents/sample_files/Txt/dracula.txt`
    - Display the **first 5 lines** of **a file**
        - `head -5 ~/Documents/sample_files/Txt/dracula.txt`
    - Display the **first 5 lines** of **multiple files**
        - `head -n 5 ~/Documents/sample_files/Txt/{dracula,war-and-peace}.txt`
    - Display the **first line of multiple files** using wildcards
        - `head -n 1 Csv/*.csv Code/*.py`
    - Display **a given number of lines** of the output of a given command
        - `ls -l ~/cis106/ |head -n 2`
    - Display the **name of the file** in the output
        - `head -v -n 7 Json/joke.json`
    - Display a **given number of bytes** instead of lines
        - `head -c 50 Txt/dracula.txt`

TAIL

- **Definition:**
    - The **tail command** displays the last N number of lines of a given file.
    - By default, it prints the **last 10 lines**. If more than one file name is provided then data from each fil e is preceded by its file name.
- **Usage:**
    - `tail + option + file(s)`
- **Examples:**

- Display the **last 10 lines** of a file
  - `tail ~/Documents/sample_files/Txt/dracula.txt`
- Display the **last 5 lines** of **a file**
  - `tail -5 ~/Documents/sample_files/Txt/dracula.txt`
- Display the **last 5 lines** of **multiple files**
  - `tail -n 5 Txt/{dracula,war-and-peace}.txt`
- Display the **last line of multiple files** using wildcards
  - `tail -n 1 Csv/*.csv Code/*.py`
- Display a **given number of lines** of the output of a given command
  - `ls -l ~/cis106/ |tail -n 2`
- Display the **name of the file** in the output
  - `tail -v -n 7 Json/joke.json`
- Display a **given number of bytes** instead of lines
  - `tail -c 50 Txt/dracula.txt`

CUT

- **Definition:**
  - The **cut command** is used to extract a specific section of each line of a file and display it to the screen.
- **Usage:**
  - `cut + option + file(s)`
- **Examples:**
  - **Display** a list of all the users in your system
    - `cut -d ':' -f1 /etc/passwd`
  - **Display** a list of all the users in your system with their **login shell**
    - `cut -d ':' -f1,7 /etc/passwd`
  - **Cut** a range of **bytes per line**
    - `cut -b 1-5 practice.txt`
  - Cut a file **using a delimiter** but changing the delimiter in the output.
    - `cut -d ':' -f1,7 --output-delimiter=' ⇒ ' /etc/passwd`
  - Cut a file **excluding a given field**
    - `cut -d ':' --complement -s -f6 /etc/passwd`
  - Cut the **permissions from the output** of ls
    - `ls -l | cut -d ' ' --complement -s -f1`

- **Note:**
  - **-d' '** : Delimiter is the character that separates the field.(here by space)
  - **-d";"** : Separate fields by ;
  - **-f3,4** : Field select the 3rd and 4th fields

- **--output-delimiter=' ⇒ '** : Replace the default space between the two fields with ⇒. **(no space after =)**
- **--complement** : Removes the fields listed and keep the rest
- **-s** : Suppress lines that don't contain the delimiter.
- **ls -l** : Produces long listing output.
- **|** : Pipe sends the output of the command on the left into the command on the right.(here output from ls is send to cut)

PASTE

- **Definition:**
  - The **paste command** is used for joining files horizontally in columns
- **Usage:**
  - `paste + option + files`
- **Examples:**
  - **Merge** two files
    - `paste users.lst codes.lst`
  - **Merge** two files **using** a different **delimiter**
    - `paste -d ":" users.lst codes.lst`

SORT

- **Definition:**
  - The **sort command** is used for sorting files. It supports sorting: alphabetically, in reverse order, by number, and by month.
  - The **sort command** follows this order unless specified otherwise:
    - Lines starting with a **number will appear before** lines starting with **letter**.
    - Lines starting with a **letter that appears earlier in the alphabet will appear before** lines starting with a **letter that appears later in the alphabet**.
    - Lines starting with a **lowercase letter will appear before** lines starting with the **same letter in uppercase**.
- **Usage:**
  - `sort + option + file`
- **Examples:**
  - Sort a file
    - `sort users.lst`
  - Sort a file and **save the output** to a new file
    - `sort -o sorted.lst users.lst`
    - sort -o **output_file input_file**
  - Sort a file in **reverse order**
    - `sort -r users.lst`
  - Sort by **column number**
    - `sort -k 2 users.lst`
  - Sort a file with **numeric data**
    - `sort -n codes.lst`

- **Check** if a file is sorted
  - `sort -c sorted.lst`
- **Sort** and **remove** duplicate entries
  - `sort -u users.lst`

WC

- **Definition:**
  - The **wc command** is used for printing the number of lines, characters and bytes in a file
- **Usage:**
  - `wc + option + file(s)`
- **Examples:**
  - Display the **number of characters** in a file
    - `wc -m dracula.txt`
  - Display the **number of lines** in a file
    - `wc -l dracula.txt`
  - Display the **number of words** in a file
    - `wc -w dracula.txt`

Notes 9

GREP

- **Definition:**
  - Grep is used to **search text in given file**. Grep works or search in a line by line basis.
- **Usage:**
  - `grep + option + search criteria + file(s)`
- **Common Options:**
  - `-i`: Enables case insensitivity (matches regardless of case).
  - `-n`: Displays line number for every line matched.
  - `-E`: Treats the pattern as an extended regular expression.
  - `-G`: Treats the pattern as a basic regular expression.
  - `-v`: Inverts the search (finds lines that do not match the pattern).
  - `-o`: Only displays the matched string.
  - `-c`: Displays the total number of times a pattern is matched.
  - `-w`: Matches only the whole word (exact pattern).
  - `-r` or `-R`: Searches recursively through directories.
- **Examples:**
  - **Search** any line that contains the word **"dracula" in the given file**.
    - `grep 'dracula' ~/Documents/sample_files/Txt/dracula.txt`
  - Search any line that contains the word "dracula" **regardless of the case**.
    - `grep -i 'dracula' ~/Documents/sample_files/Txt/dracula.txt`

- Display how many **lines** contain the **matched string**.
  - `grep -c 'Dracula' ~/Documents/sample_files/Txt/dracula.txt`
- Search any line that contains the word "dracula" **regardless of case and with number line**
  - `grep -in 'dracula' ~/Documents/sample_files/Txt/dracula.txt`
- Search for all the lines that **do not contain the word 'war'**
  - `grep -v 'war' ~/Documents/sample_files/Txt/war-and-peace.txt`
- Search and display **only the matched string** (pattern)
  - `grep -o 'pride' ~/Documents/sample_files/Txt/war-and-peace.txt`
- Display a **list of users** with the **/bin/bash login shell**
  - `grep -i "/bin/bash" /etc/passwd`
- Display your **user's information** as stored in the **/etc/passwd**
  - `grep -i $USER /etc/passwd`
- Search for a **given strings** inside files in a given directory
  - `grep -iR 'conf' /etc/`
- Search and display the **total number of times a given word appears** in a file
  - `grep -wc '/bin/bash' /etc/passwd`
- The **^ (caret) symbol** matches the empty string at the beginning of a line. Search for all the **lines that start with a given word**
  - `grep -ni '^dracula' ~/Documents/sample_files/Txt/dracula.txt`
- Search for all the **lines** that ends with the **string "nologin"**
  - `grep -n 'nologin$' /etc/passwd`
- Search for all the **lines** that **start with a capital letter**
  - `grep -n '^[A-Z]' ~/Documents/sample_files/Txt/dracula.txt`
- Search for **more than one word per line**
  - `grep -Ewn 'horror|love|scare' ~/Documents/sample_files/Txt/dracula.txt`
- Match only lines containing **IPv4 addresses**
  - `grep -E '[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}' ~/Documents/sample_files/Txt/practice.txt`
- Search all lines that contain a **character repeated 3 times**
  - `grep -E "A{3}" file.txt`
- Search all lines that contain a **phone number** of the **format 973-111-2222**
  - `grep "[[:digit:]]\{3\}[-][[:digit:]]\{3\}[-][[:digit:]]\{4\}" ~/Documents/sample_files/Csv/contacts.csv`
- Display only the options of any **command from its man page**
  - `man ls | grep "^[[:space:]]*[[:punct:]]"`
- Search for all the **lines** that contain a **single word "linux"** in all the **files in the system**
  - `grep -niR '^linux$' /`
- The **period (.) symbol** is a meta-character that **matches any single character**. It searches for all lines that contain a word starting with the letter "d" followed by any 4 characters
  - `grep -n '^d....' ~/Documents/sample_files/Txt/dracula.txt`
- **Bracket expressions** allows match a **group of characters** by enclosing them in bracket []. It matches all lines that contain the words **"list", "last", or "lost"**
  - `grep -n 'l[aio]st' ~/Documents/sample_files/Txt/dracula.txt`

AWK

- **Definition:**
  - Awk is a scripting language used for **processing and displaying text files**. Awk can work with a text file or from standard output.
  - There are several implementations of Awk: nawk, mawk, gwak, and busybox.
  - Awk performs operations line by line.
- **Usage:**
  - `awk + options + {awk command} + file + file to save (optional)`
- **Awk Variables:**

Variable	Description
$0	Whole line
$1,$2...$NF	First, second... last field
NR	Total Number of Records
NF	N number of Fields
OFS	Output Field Separator (default " ")
FS	Input Field Separator (default " ")
ORS	Output Record Separator (default "\n")
RS	Input Record Separator (default "\n")
FILENAME	Name of the file
ARGC	Number of arguments
ARGV	Array of arguments
FNR	File Number of Records
OFMT	Format for numbers (default "%.6g")
RSTART	Location in the string
RLENGTH	Length of match
SUBSEP	Multi-dimensional array separator (default "\034")
ARGIND	Argument Index
ENVIRON	Environment variables
IGNORECASE	Ignore case
CONVFMT	Conversion format

Variable	Description
ERRNO	System errors
FIELDWIDTHS	Fixed width fields

- **Examples:**
  - **Print** the **first column** of every line of a file.
    - `awk '{print $1}' ~/Documents/sample_files/Csv/cars.csv`
  - Print **first field** of /etc/passwd file
    - `awk -F: '{print $1}' /etc/passwd`
  - Print the **last field** of the /etc/passwd file
    - `awk -F: '{print $NF}' /etc/passwd`
  - Print the **first and last field** of the /etc/passwd
    - `awk -F: '{print $1," = ",$NF}' /etc/passwd`
  - Print the **first and 3 field** with line numbers
    - `awk -F: '{print NR,$1,$3}' /etc/passwd`
  - Print the **first and 4th field** with a **different field separator**
    - `awk -F: '{OFS="="}{print $1,$4}' /etc/passwd`
  - **Start printing** a file **from a given line** (exclude the first 2 lines)
    - `awk 'NR > 3 { print }' /etc/passwd`
  - **Convert** the first field **to upper/lower case**
    - `awk -F: '{print toupper($1)}' /etc/passwd`
  - Prints the **length of a line**(record)
    - `awk '{print length($0)}' /etc/passwd`
  - Print **specific fields based on a command output**. For example, the size and file name
    - `ls -lhF Documents/ | awk 'BEGIN { printf "%s\t%s\n", "Size","Name"} {print $5,"\t",$9}'`
      - BEGIN block is executed once at the start
  - Print **specific fields with a head** of the /etc/passwd file
    - `awk -F: 'BEGIN { printf "%s\t\t%s\n","User", "Shell" } {print $1,"\t",$7}' /etc/passwd`

SED

- **Definition:**
  - Sed is a **stream editor** that perform operations on **files and standard output**. For instance it can **search, find and replace,insert and deletion**.
  - By using Sed we can edit files without opening them.
- **Usage:**
  - `sed options + sed script + file`
- **Examples:**
  - **Replacing a string** in given file **globally**(replace false for true)
    - `sed 's/false/true/g' ~/Documents/sample_files/Json/joke.json`

- Replacing only the **fourth occurrence** per line in a file
  - `sed 's/false/true/4' ~/Documents/sample_files/Json/joke.json`
- Replacing from the **given number occurrence to the rest occurrences** in a file. Start at the second time the word appears and continue to till the end of the file
  - `sed 's/false/true/3g' ~/Documents/sample_files/Json/joke.json`
- Replacing string on a **specific line number**
  - `sed '8 s/false/true/' ~/Documents/sample_files/Json/joke.json`
- Replacing string on a **range of lines**
  - `sed '9,10 s/false/true/' ~/Documents/sample_files/Json/joke.json`
- **To delete** a **particular line** (line 3)
  - `sed '3d' ~/Documents/sample_files/Code/helloWorld.py`
- To delete the **last line**
  - `sed '$d' ~/Documents/sample_files/Code/helloWorld.py`
- To delete line from **range x to y**
  - `sed '2,4d' ~/Documents/sample_files/Code/helloWorld.py`
- To delete from a **given number to last line**
  - `sed '3,$d' ~/Documents/sample_files/Code/helloWorld.py`
- To delete **pattern matching line** in a file
  - `sed '/fav/d' ~/Documents/sample_files/Code/helloWorld.py`
- **To insert** one **blank line** after each line
  - `sed G ~/Documents/sample_files/Code/helloWorld.c`
- To insert **two blank lines**
  - `sed 'G;G' ~/Documents/sample_files/Code/helloWorld.c`
- **To delete** blank lines **and insert** one blank line after each line
  - `sed '/^$/d;G' ~/Documents/sample_files/Code/helloWorld.c`
- **Insert 5 spaces** to the left of every lines
  - `sed 's/^/ /' ~/Documents/sample_files/Code/helloWorld.c`

1. Explain how to use the pipe (|) for redirecting the output of a command to another.

- **Definition:**
  - The pipe allows us to **redirect** the standard output of a command to the standard input of another.
- **Usage:**
  - `command_1 | command_2 | command_3 | ----- | command_N`
- **Examples:**
  - Use **grep** to look for a string in a **particular man page**
    - `man ls | grep "human-readable"`
  - **Display** only the options of the of **any command from its man page**
    - `man ls | grep "^[[:space:]]*[[:punct:]]"`
  - Display only the **IP addresses** from the output of the ip command

        - `ip addr | grep -Eo '[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}'`
    - Display only the **2nd line in a file**
        - `head -2 ~/Documents/sample_files/Lst/users.lst | tail -1`
    - **Parse** a file with grep **and replace** a string in the output
        - `grep -i "honda" ~/Documents/sample_files/Csv/cars.csv | sed 's/Honda/tesla/g'`

2. Explain how to save the output of a command to a file (>).

- **Definition:**
    - It **redirects or save** the output of a command to a file.
- **Usage:**
    - `command output + > + file`
- **Common Options:**
    - `>`: saves standard output
    - `2>`: saves standard error
    - `&>`: savesboth output and error
- **Examples:**
    - **Save** the **output of a command** to a file
        - `ls -lA ~ > all-files-in-home.txt`
    - **Save the error** generated by a command to a file
        - `ls -lA downloads/ 2> error-of-ls`
    - Save the **error to a file** and the **success to another**
        - `ls -lA downloads/ Pictures > success.txt 2> error.txt`
    - Save the **error and success** to the **same file**
        - `ls -lA downloads/ Pictures &> alloutput.txt`
    - **Do not display errors**. Send errors to the black hole
        - `ls -lA downloads/ 2> /dev/null`

3. Explain how to append the output of a command to a file.

- **Definition:**
    - Append means to **add more to a file instead of overwriting its content**. We use **">>"** to append.
    - When we use `>` on a file that already exist and contains data, we overwrite whatever is already inside the file. For example:
        - `ls -la > allmyfiles.lst`
    - In this example, if the file allmyfiles.lst had any data prior executing the command, that data will be overwritten by the output of `ls -la`.
    - So, if we want to keep the old data? Then we use `>>`. For example:
        - `ls -la >> allmyfiles.lst`
    - This will add the output of `ls -la` to the end of the file `allmyfiles.lst`.
- **Usage:**

    - `command output + >> + file`
- **Examples:**
    - **Appends** the current directory list to the end of a file (append-example.txt).
        - `ls >> append-example.txt`
    - Appends error messages from list command to the end of a file (append-example.txt).
        - `ls Docu . 2>> append-example.txt`
    - Appends lines 1-6 from the last 7 lines of a file to another file.
        - `tail -7 append-example.txt | head -6 >> files-list.txt`