Task 1:

A Regular Expression (RegEx) is a pattern used to match text or strings. It is widely used in text-processing utilities

1. . (Dot)

Description: Matches any single character except a newline (\n).

2 ^ (Caret)

Description: Anchors the match at the beginning of a line.

3. $ (Dollar)

Description: Anchors the match at the end of a line.

4. * (Asterisk)

Descripti(Square Brackets)

on: Matches zero or more of the preceding character or pattern.

5 [ ] Description: Matches any one character inside the square brackets.

6 \ (Backslash)

Description: Escapes a special character, making it literal.
In Linux, \ is used to escape meta-characters.

7. {n} (Exact Number of Occurrences)

Description: Matches exactly n occurrences of the previous character or group.

8.? (Question Mark)

Description: Matches zero or one occurrence of the preceding character or group.

Task 2:

What are the imp features of Linux os ?

1. Open Source

The Linux kernel and most of its components are free and open source.

Users can view, modify, and distribute the source code.

2. Multitasking

Linux can run multiple processes simultaneously without interfering with each other.

3. Multiuser Support

Multiple users can use the system at the same time without affecting each other's processes or files.

4. Security

Offers features like:

- User permissions
- File encryption
- Firewall (iptables, nftables)
- SELinux/AppArmor for advanced security policies

5. Portability

Linux can run on a wide range of hardware platforms: PCs, servers, smartphones, routers, IoT devices, and supercomputers.

6. Stability and Reliability

Linux is known for its stability, often running for years without needing a reboot.

Crashes and system failures are rare compared to other operating systems.

7. Command-Line Interface (CLI) and GUI

Offers powerful CLI tools for scripting and automation.

Also supports a variety of graphical environments like GNOME, KDE, Xfce.

8. File System Support

Supports various file systems: ext4, XFS, Btrfs, FAT, NTFS, ZFS, etc.

9. Backup and Recovery

Linux supports a variety of backup and recovery tools such as:

rsync (for incremental backups)

tar (for archiving and compressing data)

dd (for disk cloning and creating disk images)

Timeshift (for system snapshots)

10.. Kernel-Based Architecture

The Linux kernel is the core component of the operating system, which manages hardware, system resources, and communication between software and hardware.

Task 3:

WHAT IS Kernal and can you explain its functions

The kernel is the core of an operating system (OS). It is the lowest level of software that directly interacts with the computer's hardware, such as the CPU, memory, storage devices, and input/output devices. The kernel manages system resources and provides an interface between the hardware and the higher-level software (like applications or the user interface).

Functions

Managing Processes

- What it does: The kernel decides which programs (processes) run, when they run, and for how long.
- Example: If you open a web browser and a text editor, the kernel makes sure both run smoothly and share the computer's CPU time.

Memory Management

- What it does: The kernel manages the computer's memory (RAM), making sure each program gets the space it needs without interfering with others.
- Example: If you open several programs, the kernel gives each one its own space in memory and swaps data in and out of RAM when needed.

Device Management

What it does: The kernel helps the operating system communicate with hardware devices like your keyboard, mouse, printer, and hard drive.

Example: When you print a document, the kernel tells the printer what to do using a driver.

File System Management

What it does: The kernel organizes how files are stored on your hard drive or SSD, and manages file access.

Example: When you open a file, the kernel finds it on the disk and shows it to you.

Task 4:

What is BASH? Full form with explaination.

5min 10.00 to 10.05

BASH stands for Bourne Again Shell, and it is a command-line shell and scripting language used to interact with your computer's operating system.

- It allows users to type commands to manage files, run programs, and automate tasks.
- BASH also allows you to write scripts (small programs) to automate repetitive tasks.

It is the default shell on most Linux systems and macOS (prior to Catalina).

In simpler terms, BASH is a tool for controlling your computer through text-based commands.

Task 5:

What is the diffrenece between window and linux

Linux:

Linux is an open-source operating system, which means its source code is freely available for anyone to use, modify, and distribute. It's highly customizable, often used by developers, and works well on both personal computers and servers. Popular distributions (or versions) of Linux include Ubuntu, Debian, and Fedora.

Windows:

Windows is a proprietary operating system created by Microsoft. Unlike Linux, it is not open-source and is typically paid. It is widely used on personal computers and laptops due to its user-friendly interface, support for a wide range of software, and popularity in both home and business environments.

Key Difference:

Linux is open-source, customizable, and often used by developers and servers.

Windows is proprietary, user-friendly, and commonly used for personal computers.

Task 6:

Define the basic components of Linux

1. Kernel

The core of the Linux operating system.

Manages hardware resources like CPU, memory, and peripheral devices.

Controls communication between the hardware and software.

Handles process management, memory management, device drivers, file system management, and system calls.

The kernel runs in privileged mode (kernel space), with direct access to hardware.

2. System Library

A set of functions that programs can use to perform common tasks without interacting with the kernel directly.

For example, the GNU C Library (glibc) is one of the most widely used libraries, providing standard functions for file handling, input/output, etc.

Libraries simplify development by offering an interface to system resources and services.

3. System Utilities

Tools and programs that perform specific tasks like managing files, user management, process management, etc.

Examples include commands like `ls`, `cp`, `rm`, `ps`, `top`, `df`, and more.

These utilities are designed to allow users and administrators to interact with the system.

4. Shell

The command-line interface (CLI) that allows users to interact with the system.

The shell interprets and executes user commands, often providing scripts to automate tasks.

Popular shells include Bash (Bourne Again Shell), Zsh, Fish, and others.

It acts as an intermediary between the user and the kernel.

5. File System

Organizes how data is stored and retrieved on storage devices.

Linux uses a hierarchical file system structure, with the root directory (`/`) at the top.

Common file systems in Linux include ext4, XFS, Btrfs, etc.

It includes system files, user files, and directories (like `/home`, `/var`, `/usr`).

6. Device Drivers

Special programs that allow the operating system to communicate with hardware devices (e.g., printers, network cards, hard drives, etc.).

Linux includes a wide range of built-in drivers, and additional drivers can be installed as needed.

7. User Interface

Linux can provide graphical user interfaces (GUIs) or can run in text-based mode.

X Window System (X11) is the foundational layer for GUIs on Linux, providing the framework for graphical environments like GNOME, KDE, etc.

GUI environments offer a more user-friendly way to interact with the system, but the command line is often preferred for its power and flexibility.

8. Processes

Every running program is a process in Linux.

The kernel manages process scheduling, which determines how processes are executed, when they run, and how resources are allocated to them.

Processes are assigned a Process ID (PID), and each process can have its own memory space and can be managed individually.

Task 7:

Is it legal to edit Kernal?

The term **"edit"** or **"modifying"** the kernel refers to making changes or adjustments to the source code of the Linux kernel. This can involve:

1. **Changing existing code**: Editing functions, algorithms, or components within the kernel to enhance performance, fix bugs, or add new features.

2. **Adding new code**: Introducing new modules or components to the kernel, such as device drivers, file system support, or security enhancements.

3. **Rebuilding the kernel**: After making changes to the source code, compiling and rebuilding the kernel so that it can be used on a system.

The kernel is the core part of the operating system that interacts directly with hardware and manages system resources. Modifying it allows users to tailor the behavior of the system to their specific needs or preferences.

Legally, modifying the kernel is allowed under the **GNU General Public License (GPL) version 2**, which governs the Linux kernel's distribution and usage. However, there are legal obligations regarding the redistribution of modified kernels, such as making the modified source code available to others and ensuring the modified code is also distributed under the same GPL license.

LILO (Linux Loader) is a **bootloader** for Linux operating systems. It is a program that is responsible for loading the **Linux kernel** into memory during the system's boot process and transferring control of the system to the operating system.

Key Points:

- **Bootloader**: A small program that runs when the system is powered on and loads the operating system into memory.

- **Primary Role**: LILO loads the Linux kernel (or other operating systems) into memory and starts the boot process.

- **Configuration**: LILO is configured using a file called `lilo.conf`, where settings such as kernel image locations and boot parameters are specified.

- **Dual Boot**: LILO can also be used to set up **dual-boot** systems, allowing users to choose between multiple operating systems (e.g., Linux and Windows) at startup.

- **Manual Updates**: Unlike more modern bootloaders like GRUB, LILO requires the user to manually update the bootloader after any changes to the kernel.

LILO was widely used in the past but has largely been replaced by GRUB in most modern Linux distributions due to its more flexible and automatic configuration.

Task 8:

Can you explain LILO

LILO (Linux Loader) is a bootloader for Linux operating systems. It is a program that is responsible for loading the Linux kernel into memory during the system's boot process and transferring control of the system to the operating system.

Key Points:

- Bootloader: A small program that runs when the system is powered on and loads the operating system into memory.

- Primary Role: LILO loads the Linux kernel (or other operating systems) into memory and starts the boot process.

- Configuration: LILO is configured using a file called `lilo.conf`, where settings such as kernel image locations and boot parameters are specified.

- Dual Boot: LILO can also be used to set up dual-boot systems, allowing users to choose between multiple operating systems (e.g., Linux and Windows) at startup.

- Manual Updates: Unlike more modern bootloaders like GRUB, LILO requires the user to manually update the bootloader after any changes to the kernel.

LILO was widely used in the past but has largely been replaced by GRUB in most modern Linux distributions due to its more flexible and automatic configuration.

Task 9:

What is shell? How many shells are there and what are they ? can you explain.

A shell is a command-line interface (CLI) in an operating system that allows users to interact with the system by typing commands. It acts as an intermediary between the user and the operating system, allowing users to execute commands, manage files, run scripts, and configure system settings.

In simpler terms, the shell is a program that reads your commands (entered via the command line), interprets them, and then executes the corresponding action, such as running a program or interacting with the system's files.

The shell can either be text-based (command line) or graphical (GUI), but the most common usage in Linux is through the text-based command-line interface.

Types of Shells

There are several types of shells in Linux and Unix-based operating systems. These shells are command interpreters that differ in terms of features, syntax, and capabilities. Here's a look at some of the most common ones:

---

1. Bash (Bourne Again Shell)

- **Most Popular Shell: Bash is the default shell for most Linux distributions and macOS.**
- **Features: It is a hybrid shell that incorporates features from the Bourne shell (sh), C shell (csh), and others.**
- **Capabilities:**
    - **Command-line editing.**
    - **Job control (running processes in the background).**
    - **Supports scripts and functions.**
    - **Customizable via `~/.bashrc`.**
- **Usage: It's widely used for interactive and script-based tasks. It's powerful and flexible for both beginners and advanced users.**

2. Sh (Bourne Shell)

- **Older Shell: The Bourne Shell (sh) is one of the oldest shells and served as the basis for many other shells.**
- **Features:**
    - **Basic scripting and command execution.**
    - **It is simpler and lacks some of the modern features found in newer shells.**
    - **POSIX-compliant: The Bourne shell is the standard shell defined by POSIX (Portable Operating System Interface).**
- **Usage: It's often used in script files (sh scripts) and for compatibility with older systems.**

3. Zsh (Z Shell)

- **Advanced Shell: Zsh is an extension of the Bourne shell with additional features like improved tab completion, auto-corrections, and better customization options.**
- **Features:**
    - **Autocompletion: Enhanced tab completion for file names, commands, and arguments.**
    - **Customization: Supports themes, plugins, and is highly configurable.**
    - **Scripting: Like Bash, Zsh is also useful for writing shell scripts.**
    - **Interactive Use: Very popular among power users for its user-friendly features.**
- **Usage: It's often preferred by developers and power users who need advanced features for command-line productivity.**

4. Fish (Friendly Interactive Shell)

- **User-Friendly: Fish is a user-friendly, modern shell designed for ease of use.**

- **Features:**
    - **Syntax highlighting: It highlights valid and invalid commands.**
    - **Autosuggestions: It suggests commands based on history and syntax.**
    - **Tab completion: Advanced tab completion with more context.**
- **Usage: Best for users who want an interactive and intuitive experience without needing to configure a lot of settings. It's often used in development environments.**

5. Csh (C Shell)

- **C-Like Syntax: The C shell (csh) is a shell that has a syntax similar to the C programming language.**
- **Features:**
    - **Supports aliases and scripting.**
    - **Job control for managing running processes.**
    - **Built-in history management.**
- **Usage: While it's not as popular as Bash or Zsh today, it's sometimes used by programmers who prefer C-style syntax.**

6. Tcsh (TENEX C Shell)

- **Enhanced Csh: Tcsh is an improved version of the C shell, adding features such as command-line editing and filename completion.**
- **Features:**
    - **Command-line editing.**
    - **Filename completion.**
    - **History expansion.**

- **Usage: It is typically used in systems that rely on the C-shell family but with enhanced features for usability.**

7. Ksh (Korn Shell)

- **Extended Bourne Shell: The Korn Shell (ksh) is an extended version of the Bourne shell, combining features from the C shell as well as new innovations.**
- **Features:**
    - **Supports functions and scripts with advanced control structures.**
    - **Job control, command history, and aliases.**
    - **POSIX-compliant.**
- **Usage: It's popular in both interactive and scripting environments, particularly for system administration and Unix-based programming.**

Task 10:

What is Swap space ?

Swap space in Linux is like an overflow for your computer's memory (RAM).

> When your RAM is full, Linux moves some data to the swap space on your hard drive or SSD to free up RAM.

This helps keep your system running smoothly, but swap is slower than RAM because it's on the hard drive.

In short, swap is extra space to prevent your computer from running out of memory.

Task 11:

What is Mount ? how do you mount and unmount file system in Linux?

Mount: Mounting is the process of making a storage device (like a hard drive, USB stick, or CD/DVD) accessible to your system by linking it to a directory (called a **mount point**) in the Linux file system. After mounting, you can access the files on the device through the mount point.

Unmount: Unmounting is the opposite of mounting. It means detaching a device from the file system, making it no longer accessible through the mount point. It's important to unmount a device properly before removing it to avoid data loss or file corruption.

o mount a device, you use the `mount` command

. sudo mount <device> <mount_point>

`<device>` is the storage device (like `/dev/sdb1`, `/dev/cdrom`, etc.).

`<mount_point>` is the directory where you want the device to be accessible (e.g., `/mnt/usb`, `/media/usb`, etc.).

Example:

If you want to mount a USB drive:

```
sudo mount /dev/sdb1 /mnt/usb
```

This mounts the device /dev/sdb1 to the directory /mnt/usb. You can now access the USB files by going to /mnt/usb.

How to Unmount a File System in Linux?

To unmount a device, you use the `umount` command.

sudo umount <device>

`<device>` is the device name (like `/dev/sdb1`).

sudo umount /dev/sdb1

Task 12:

What is chmod command ? how to use it?

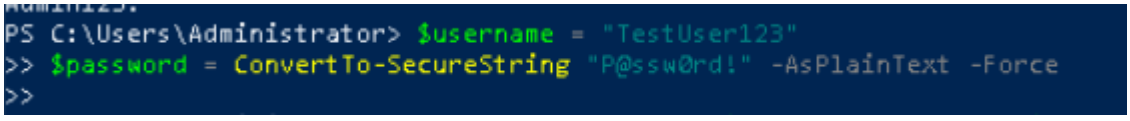
The `chmod` (change mode) command in Linux is used to modify the permissions of files or directories. It controls who can read, write, or execute a file or directory.

Linux permissions are categorized into three groups:

- User (u): The file's owner.
- Group (g): The group that the file belongs to.
- Others (o): Everyone else who is not the owner or in the group.

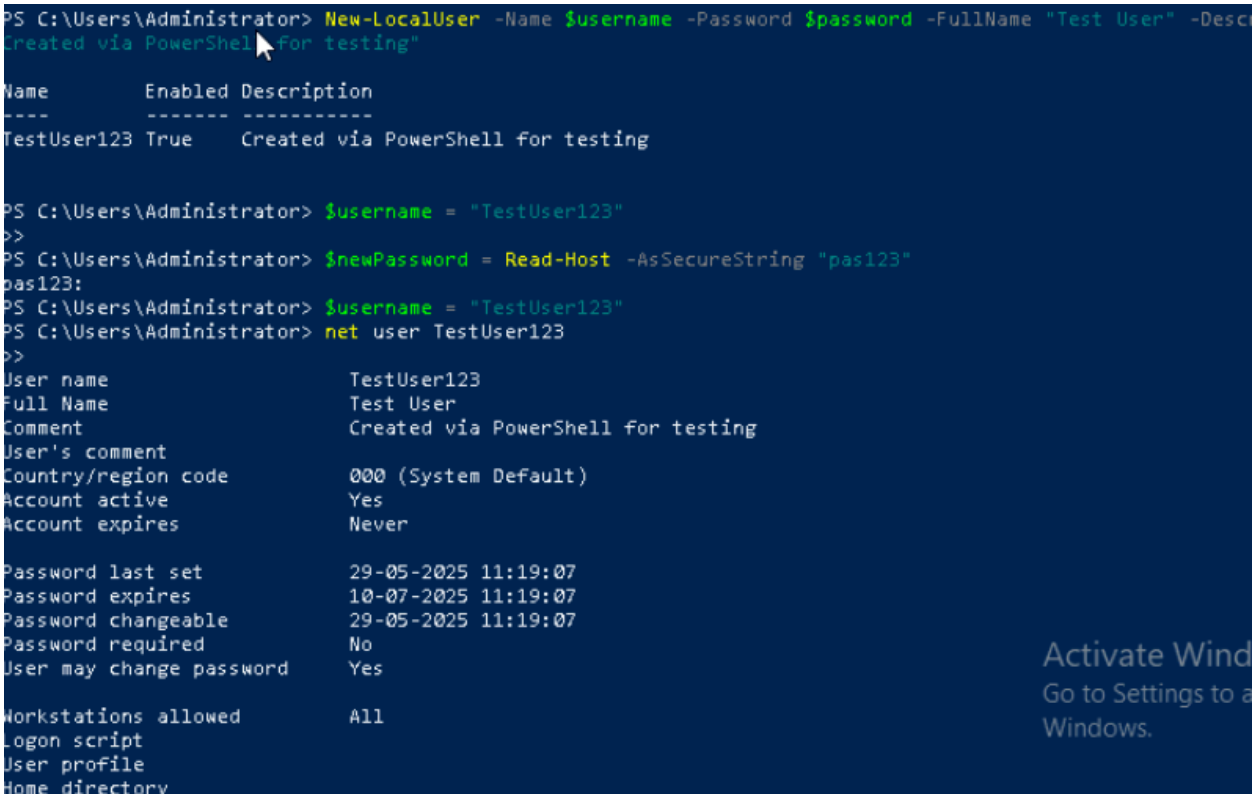
Task 13:

Can you add a new user account? Crate a new user in different ways and paste ss



Task 14:

Can you change the password of a user?
How do you do that? Plz share ss



Task 15:

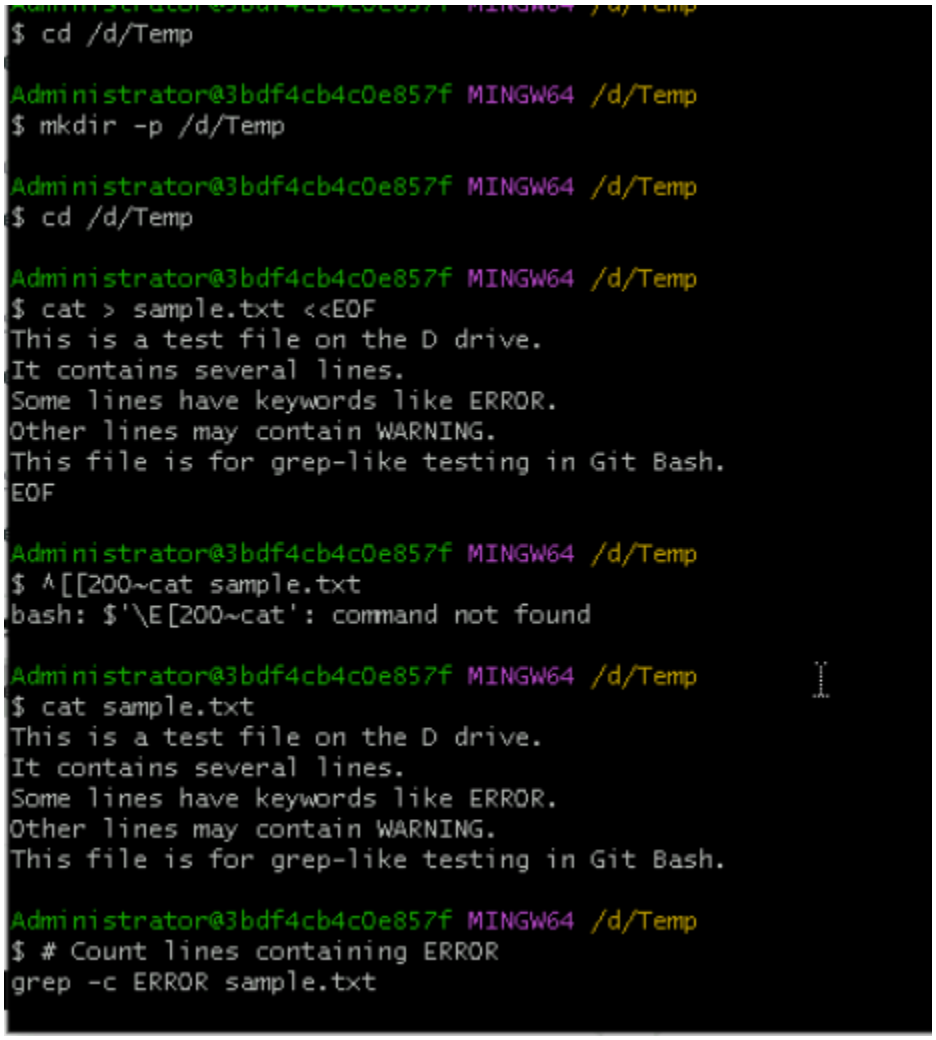
What is diff between Process and Thread?

Process:
A **process** is an **independent program** that is running on a computer. It is a self-contained execution unit that has its own **memory space**, **code**, **data**, and **system resources**. Each process operates independently from others and can run concurrently with other processes. Processes are isolated from each other, meaning one process cannot directly access the memory or resources of another process.

Thread:
A **thread** is the smallest **unit of execution** within a process. A thread is sometimes called a "lightweight process" because it shares the **same memory space** and **resources** as other threads within the same process. Threads are used to perform multiple tasks concurrently within a single process. Each thread has its own **execution context** (like a program counter, registers, and stack), but all threads share the same heap and global variables.

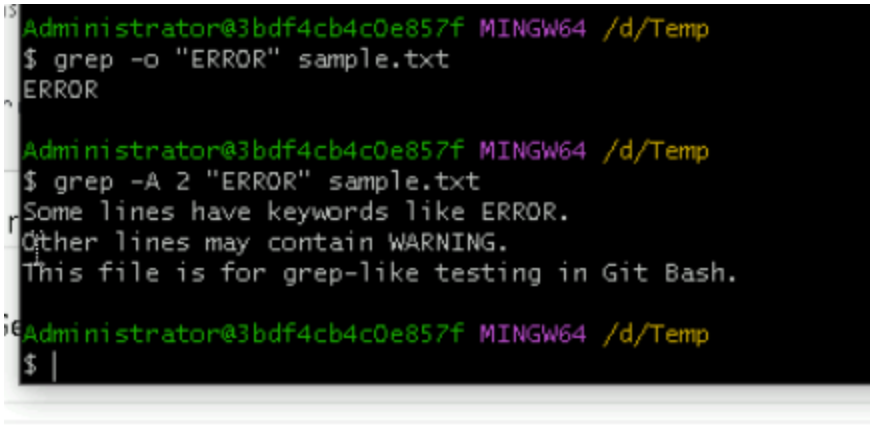
Task 16:

Linux Grep commands .. plz work on it..



```
Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ cat sample.txt
This is a test file on the D drive.
It contains several lines.
Some lines have keywords like ERROR.
Other lines may contain WARNING.
This file is for grep-like testing in Git Bash.

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ # Count lines containing ERROR
grep -c ERROR sample.txt

1

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -i error sample.txt
Some lines have keywords like ERROR.

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -n ERROR sample.txt
3:Some lines have keywords like ERROR.

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -v ERROR sample.txt
This is a test file on the D drive.
It contains several lines.
Other lines may contain WARNING.
This file is for grep-like testing in Git Bash.

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -v ERROR sample.txt
This is a test file on the D drive.
It contains several lines.
Other lines may contain WARNING.
This file is for grep-like testing in Git Bash.
```

```
Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ ^[[200~grep -A 3 ERROR sample.txt~
bash: $'\E[200~grep': command not found

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -A 3 ERROR sample.txt
Some lines have keywords like ERROR.
Other lines may contain WARNING.
This file is for grep-like testing in Git Bash.

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ cat > patterns.txt <<EOF
ERROR
WARNING
EOF

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -f patterns.txt sample.txt
Some lines have keywords like ERROR.
Other lines may contain WARNING.

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -E "ERROR|WARNING" sample.txt
Some lines have keywords like ERROR.
Other lines may contain WARNING.

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -E
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.

Administrator@3bdf4cb4c0e857f MINGW64 /d/Temp
$ grep -o "ERROR" sample.txt
ERROR
```



Task 17 AWK commands in doc 15 Linux AWK commands..

13.30 to 14.30 lunch break

10 done out of 29

14.31 to 14.36 to 14.45→ for AWK commands

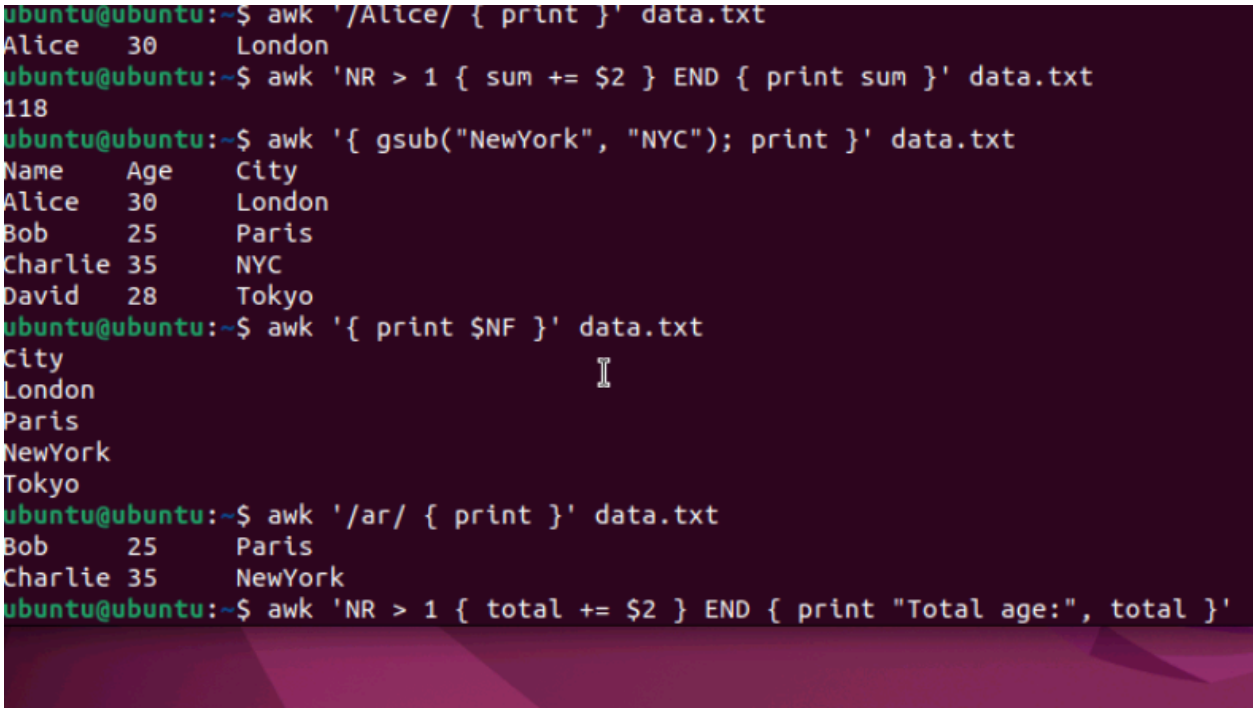
This is the link to open ODs files in linux.. You need to download the compatibles
https://www.adobe.com/uk/acrobat/resources/document-files/open-doc/ods.html

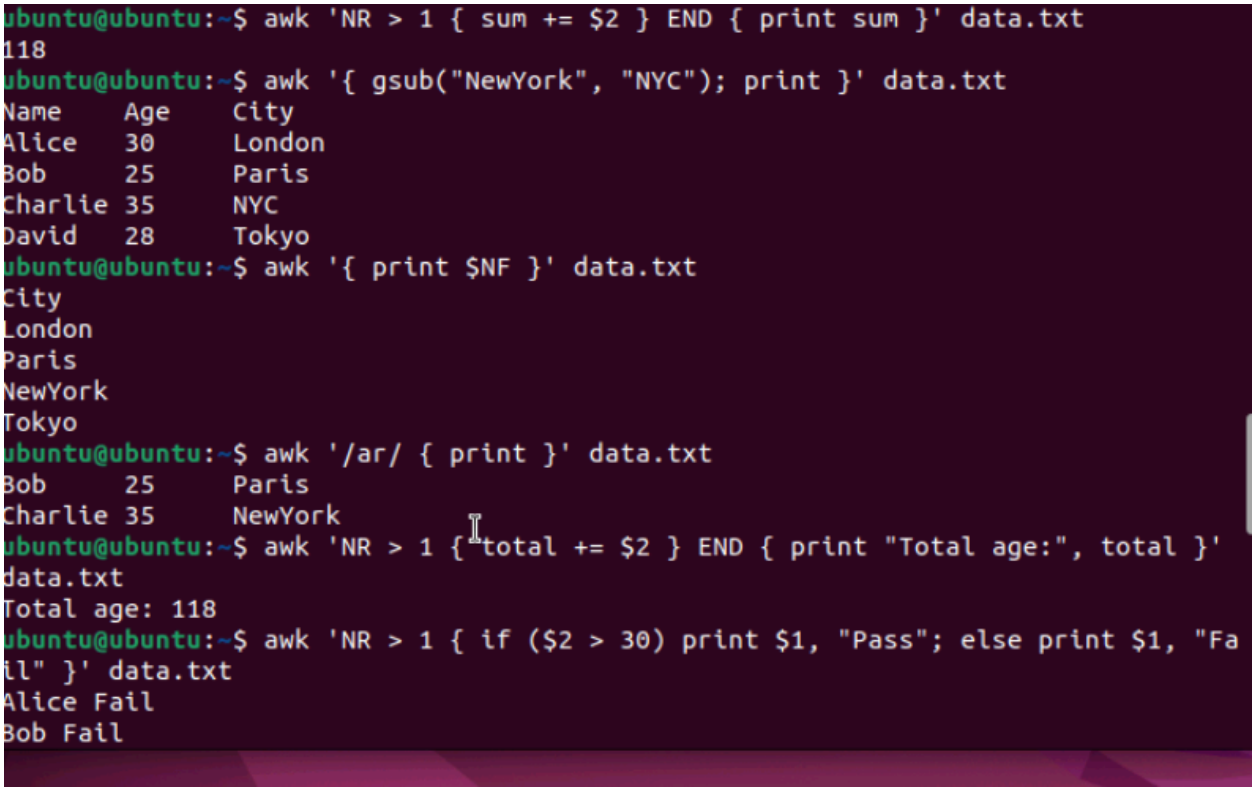
```
ubuntu@ubuntu:~$ cat > data.txt <<EOF
Name    Age     City
Alice   30      London
Bob     25      Paris
Charlie 35      NewYork
David   28      Tokyo
EOF
ubuntu@ubuntu:~$ cat data.txt
Name    Age     City
Alice   30      London
Bob     25      Paris
Charlie 35      NewYork
David   28      Tokyo
ubuntu@ubuntu:~$ awk '{print}' data.txt
Name    Age     City
```

```
ubuntu@ubuntu:~$ awk '{print}' data.txt
Name    Age     City
Alice   30      London
Bob     25      Paris
Charlie 35      NewYork
David   28      Tokyo
ubuntu@ubuntu:~$ awk '{print $1}' data.txt
Name
Alice
Bob
Charlie
David
ubuntu@ubuntu:~$ awk '{print $2}' data.txt
Age
30
25
35
28
ubuntu@ubuntu:~$ awk 'NR > 1 {print $1, $2}' data.txt
Alice 30
Bob 25
Charlie 35
David 28
```

```
ubuntu@ubuntu:~$ awk 'NR > 1 {print $1, $2}' data.txt
Alice 30
Bob 25
Charlie 35
David 28
ubuntu@ubuntu:~$ cat -A data.txt
Name    Age     City$
Alice   30      London$
Bob     25      Paris$
Charlie 35      NewYork$
David   28      Tokyo$
ubuntu@ubuntu:~$ awk -F '[ \t]+' 'NR > 1 && $2 > 28 {print $1, $2}' data.txt
Alice 30
Charlie 35
ubuntu@ubuntu:~$ awk '{print NR, $1, $2, $3}' data.txt
1 Name Age City
2 Alice 30 London
3 Bob 25 Paris
4 Charlie 35 NewYork
5 David 28 Tokyo
```

```
ubuntu@ubuntu:~$ awk -F '[ \t]+' 'NR > 1 && $2 > 28 {print $1, $2}' data.txt
Alice 30
Charlie 35
ubuntu@ubuntu:~$ awk '{print NR, $1, $2, $3}' data.txt
1 Name Age City
2 Alice 30 London
3 Bob 25 Paris
4 Charlie 35 NewYork
5 David 28 Tokyo
ubuntu@ubuntu:~$ awk '{print $1,$3 } data.txt
>
> awk '{print $1,$3 } data.txt
awk: 1: unexpected character '.'
awk: line 4: missing } near end of file
ubuntu@ubuntu:~$ awk '{print $1, $3}' data.txt
Name City
Alice London
Bob Paris
Charlie NewYork
David Tokyo
ubuntu@ubuntu:~$ awk '/error/ {print}' data.txt
ubuntu@ubuntu:~$ awk '/error/ { print }' data.txt
ubuntu@ubuntu:~$ awk '/Alice/ { print }' data.txt
Alice   30      London
```


ubuntu@ubuntu:~$ awk '/Alice/ { print }' data.txt
Alice 30 London
ubuntu@ubuntu:~$ awk 'NR > 1 { sum += $2 } END { print sum }' data.txt
118
ubuntu@ubuntu:~$ awk '{ gsub("NewYork", "NYC"); print }' data.txt
Name Age City
Alice 30 London
Bob 25 Paris
Charlie 35 NYC
David 28 Tokyo
ubuntu@ubuntu:~$ awk '{ print $NF }' data.txt
City
London
Paris
NewYork
Tokyo
ubuntu@ubuntu:~$ awk '/ar/ { print }' data.txt
Bob 25 Paris
Charlie 35 NewYork
ubuntu@ubuntu:~$ awk 'NR > 1 { total += $2 } END { print "Total age:", total }'


ubuntu@ubuntu:~$ awk 'NR > 1 { sum += $2 } END { print sum }' data.txt
118
ubuntu@ubuntu:~$ awk '{ gsub("NewYork", "NYC"); print }' data.txt
Name Age City
Alice 30 London
Bob 25 Paris
Charlie 35 NYC
David 28 Tokyo
ubuntu@ubuntu:~$ awk '{ print $NF }' data.txt
City
London
Paris
NewYork
Tokyo
ubuntu@ubuntu:~$ awk '/ar/ { print }' data.txt
Bob 25 Paris
Charlie 35 NewYork
ubuntu@ubuntu:~$ awk 'NR > 1 { total += $2 } END { print "Total age:", total }'
data.txt
Total age: 118
ubuntu@ubuntu:~$ awk 'NR > 1 { if ($2 > 30) print $1, "Pass"; else print $1, "Fa
il" }' data.txt
Alice Fail
Bob Fail

```
ubuntu@ubuntu:~$ awk 'NR > 1 { if ($2 > 30) print $1, "Pass"; else print $1, "Fa
il" }' data.txt
Alice Fail
Bob Fail
Charlie Pass
David Fail
ubuntu@ubuntu:~$ awk 'BEGIN { FS=" "; OFS=" | " } { print $1, $2, $3 }' data.txt

Name | Age | City
Alice | 30 | London
Bob | 25 | Paris
Charlie | 35 | NewYork
David | 28 | Tokyo
ubuntu@ubuntu:~$ awk 'NR > 1 { if ($2 > 30) print $1, "Pass"; else print $1, "Fa
il" }' data.txt
Alice Fail
Bob Fail
Charlie Pass
David Fail
ubuntu@ubuntu:~$ awk 'NR > 1; function square(x) { return x*x } { print $1, squa
re($2) }' data.txt
Name 0
```

```
ubuntu@ubuntu:~$ awk 'NR > 1; function square(x) { return x*x } { print $1, squa
re($2) }' data.txt
Name 0
Alice   30      London
Alice 900
Bob     25      Paris
Bob 625
Charlie 35      NewYork
Charlie 1225
David   28      Tokyo
David 784
```

Task 18:

How to check file access permission in Linux?

Hint use:
Ls -l

```
ubuntu@ubuntu:~$ ls -l data.txt
-rw-rw-r-- 1 ubuntu ubuntu 107 May 29 14:24 data.txt
```

Task 19:

What are the default permissions for a new file ?

Plz find out for

Owner → ?
Group → ?
All and others → ?

```
ubuntu@ubuntu:~$ chmod 640 hi.txt
ubuntu@ubuntu:~$ chmod 444 hi.txt
ubuntu@ubuntu:~$ ls -l hi.txt
-r--r--r-- 1 ubuntu ubuntu 4 May 29 15:32 hi.txt
```

Task 20:

What is the command to change the permisssion to read only for the owner, group and all other users

Hlnt: chmod 444 filename

```
ubuntu@ubuntu:~$ chmod 444 hi.txt
ubuntu@ubuntu:~$ ls -l hi.txt
-r--r--r-- 1 ubuntu ubuntu 4 May 29 15:32 hi.txt
ubuntu@ubuntu:~$ chmod 755 chmod.exercises
```

Task 21:

Can you change the file permissions to match the following:

- **owner:** `Read` **and** `Write`
- **group:** `Read`
- **other: no permissions (**`None`**)**

```
ubuntu@ubuntu:~$ chmod 640 hi.txt
ubuntu@ubuntu:~$ ls -l hi.txt
-rw-r----- 1 ubuntu ubuntu 4 May 29 15:32 hi.txt
ubuntu@ubuntu:~$
```

Task 22:

What was the command for changing the file permissions to -rw-r-----?

Hint : use chmod 640 filename

```
ubuntu@ubuntu:~$ chmod 640 hi.txt
ubuntu@ubuntu:~$ ls -l hi.txt
-rw-r----- 1 ubuntu ubuntu 4 May 29 15:32 hi.txt
ubuntu@ubuntu:~$
```

Task 23:

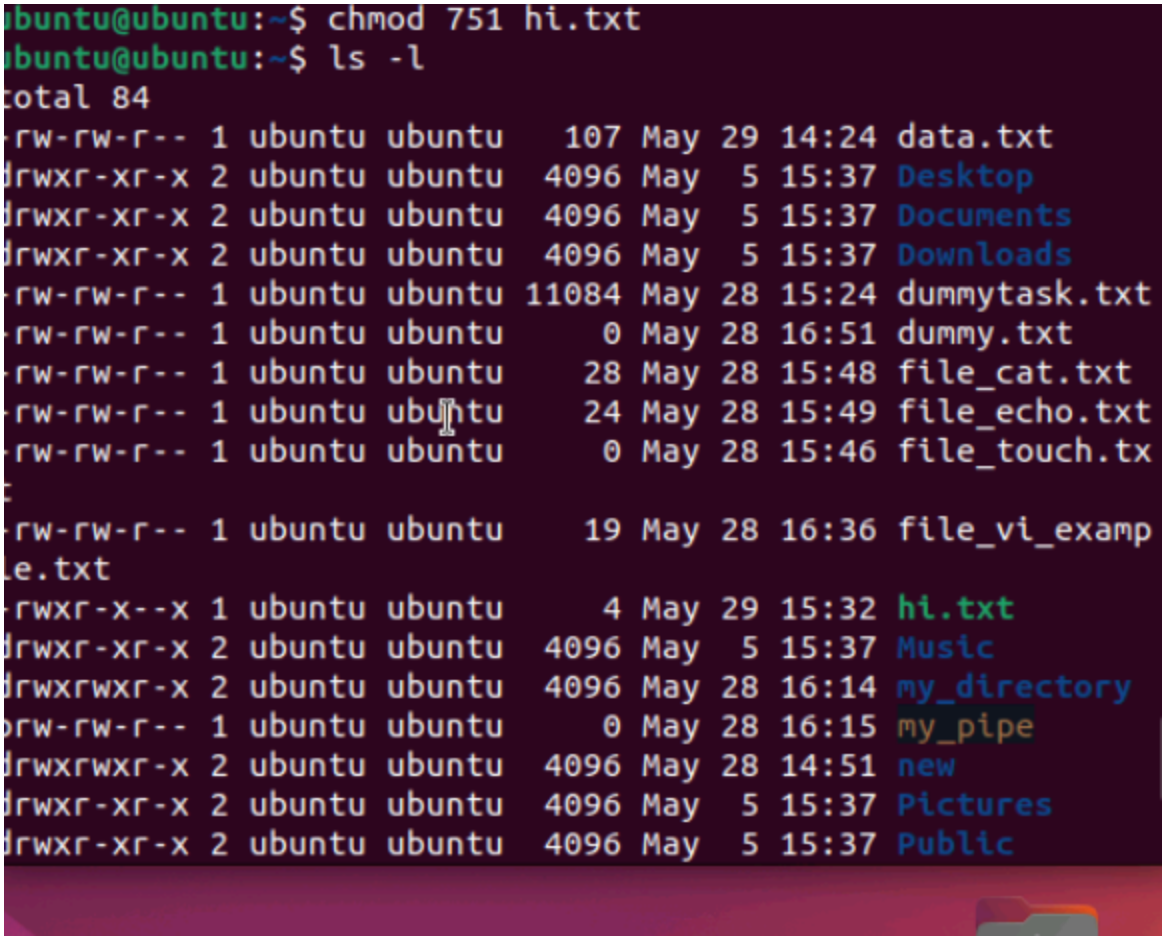
Change chmod.exercises permissions to -rwxr-x--x

Change the file permissions to match the following:

owner: Read, Write and Execute

group: Read and Execute

other: Execute



Task 24:

What was the command for changing the file permissions to -rwxr-x--x

```
ubuntu@ubuntu:~$ chmod 711 hi.txt
ubuntu@ubuntu:~$ ls -l
total 84
-rw-rw-r-- 1 ubuntu ubuntu   107 May 29 14:24 data.txt
drwxr-xr-x 2 ubuntu ubuntu  4096 May  5 15:37 Desktop
drwxr-xr-x 2 ubuntu ubuntu  4096 May  5 15:37 Documents
drwxr-xr-x 2 ubuntu ubuntu  4096 May  5 15:37 Downloads
-rw-rw-r-- 1 ubuntu ubuntu 11084 May 28 15:24 dummytask.txt
-rw-rw-r-- 1 ubuntu ubuntu     0 May 28 16:51 dummy.txt
-rw-rw-r-- 1 ubuntu ubuntu    28 May 28 15:48 file_cat.txt
-rw-rw-r-- 1 ubuntu ubuntu    24 May 28 15:49 file_echo.txt
-rw-rw-r-- 1 ubuntu ubuntu     0 May 28 15:46 file_touch.tx
t
-rw-rw-r-- 1 ubuntu ubuntu    19 May 28 16:36 file_vi_examp
le.txt
-rwx--x--x 1 ubuntu ubuntu     4 May 29 15:32 hi.txt
```

Task 25:

Guys what will this command do?

```
chown -c master file1.txt
```

```
buntu@ubuntu:~$ sudo chown -c meeraj hi.txt
hanged ownership of 'hi.txt' from ubuntu to meeraj
buntu@ubuntu:~$
```

Task 26

Can you define what is a process

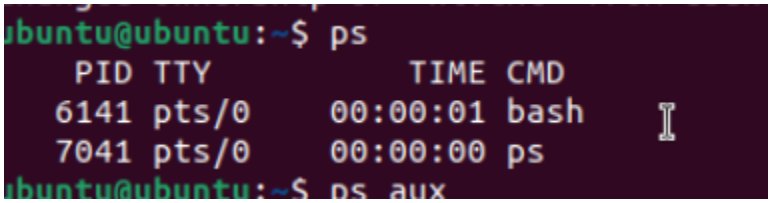
What is command to check foreground process and background process

A **process** in a computer system, particularly in an operating system like Linux, refers to an instance of a program that is currently being executed. It consists of the program code (called the text section), its current activity (including the program counter, processor registers, etc.), and the resources allocated to it, such as memory, input/output devices, and files.

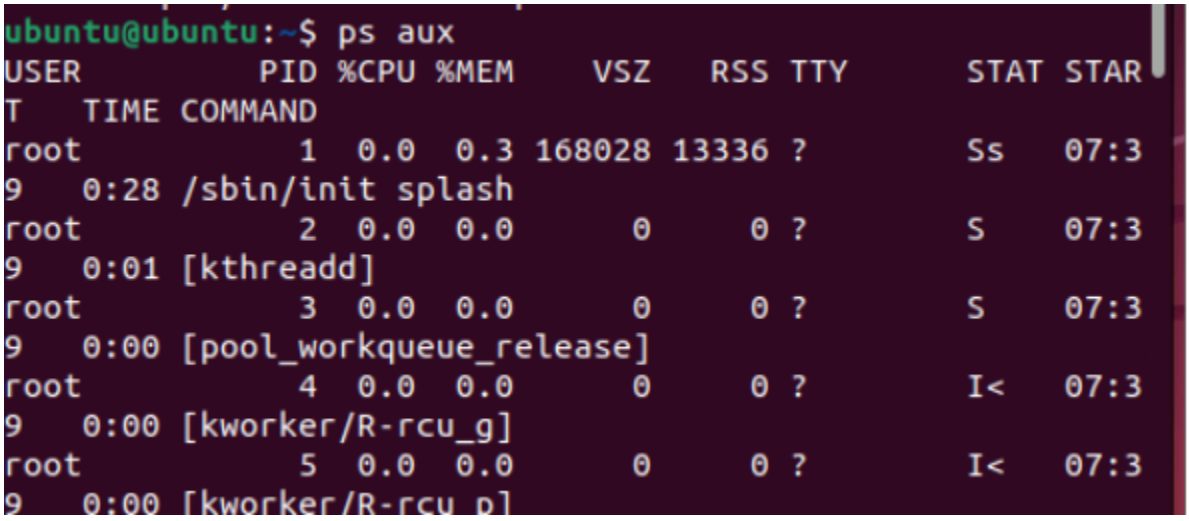
In Linux, every process is identified by a **Process ID (PID)**, a unique number assigned to it by the operating system when it starts. Processes are fundamental units of execution in an OS, and the OS manages their execution, scheduling, and resources.

To view running foreground processes:

The `ps` command is helpful to see the processes running under your user account



Or to see processes with more details, you can use:



Task 29:

What will ps -f command do ? plz try n check .. ss required.

What will ps -f command do ? plz try n check .. ss required.

The `ps` command in Linux is used to display information about running processes. The `-f` option (which stands for **full format**) provides **detailed information** about the processes. Specifically, it shows the processes in a "full" format with extra columns such as:

```
ubuntu@ubuntu:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
ubuntu      6141    6082  0 14:24 pts/0    00:00:01 bash
ubuntu      7047    6141  0 17:01 pts/0    00:00:00 ps -f
```

Task 28

Can you list all the running processes?

```
ubuntu@ubuntu:~$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT STAR
T   TIME COMMAND
root           1  0.0  0.3 168028 13336 ?        Ss   07:3
9   0:28 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    07:3
9   0:01 [kthreadd]
root           3  0.0  0.0      0     0 ?        S    07:3
9   0:00 [pool_workqueue_release]
root           4  0.0  0.0      0     0 ?        I<   07:3
9   0:00 [kworker/R-rcu_g]
root           5  0.0  0.0      0     0 ?        I<   07:3
9   0:00 [kworker/R-rcu_p]
root           6  0.0  0.0      0     0 ?        I<   07:3
9   0:00 [kworker/R-slub_]
root           7  0.0  0.0      0     0 ?        I<   07:3
```

Task 27:

What is command to check foreground process and background process

Lets play with shell variables

Task 30:

Can you createa a variable name with your name in it

Ex:

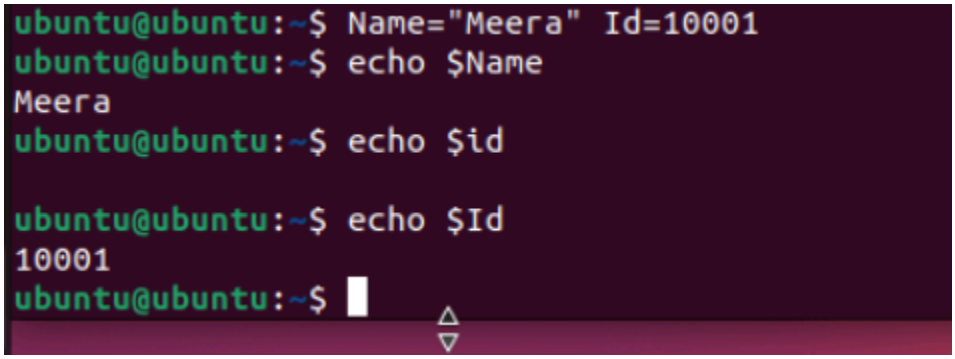
Name = “prasunamba”

Id = 10001

And check

Echo $Name

Chek the output



Task 31:

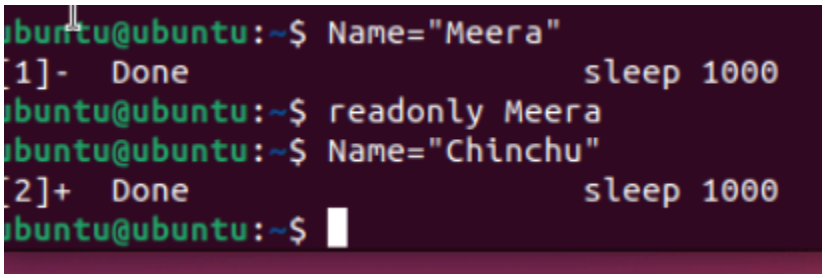
Can you make the above name variable read only..

Ex:

Name = “Prasunamba”

Readonly Name

Name = “Meher” —>what will this display.. Is it saying read only?? Pl check



Task 32:

Now will unset or delete the variables

Use the below command and check

Unset Name

Now check for

echo $Name —> this should not print anything.. Plz try also specify the reason

Read n Know that

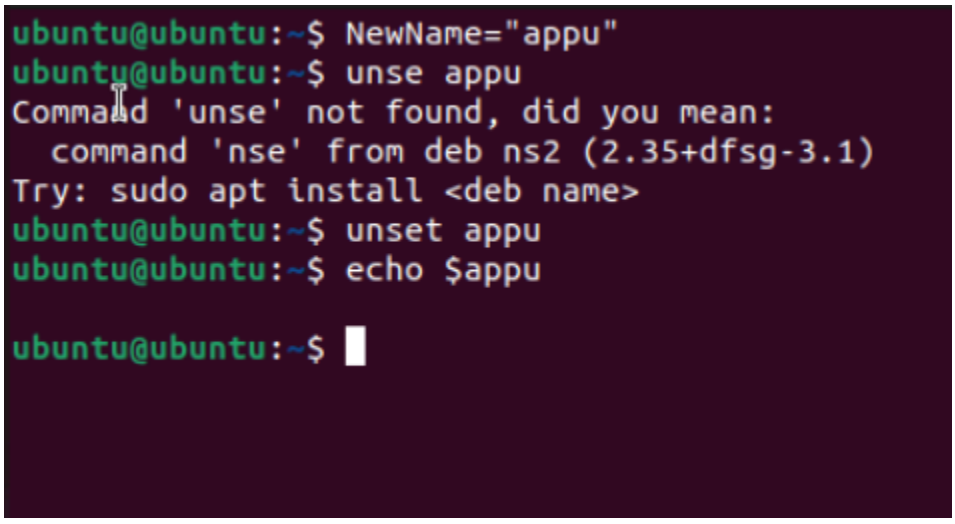
Variable Types

When a shell is running, three main types of variables are present −

Local Variables − A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at the command prompt.

Environment Variables − An environment variable is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually, a shell script defines only those environment variables that are needed by the programs that it runs.

Shell Variables − A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables

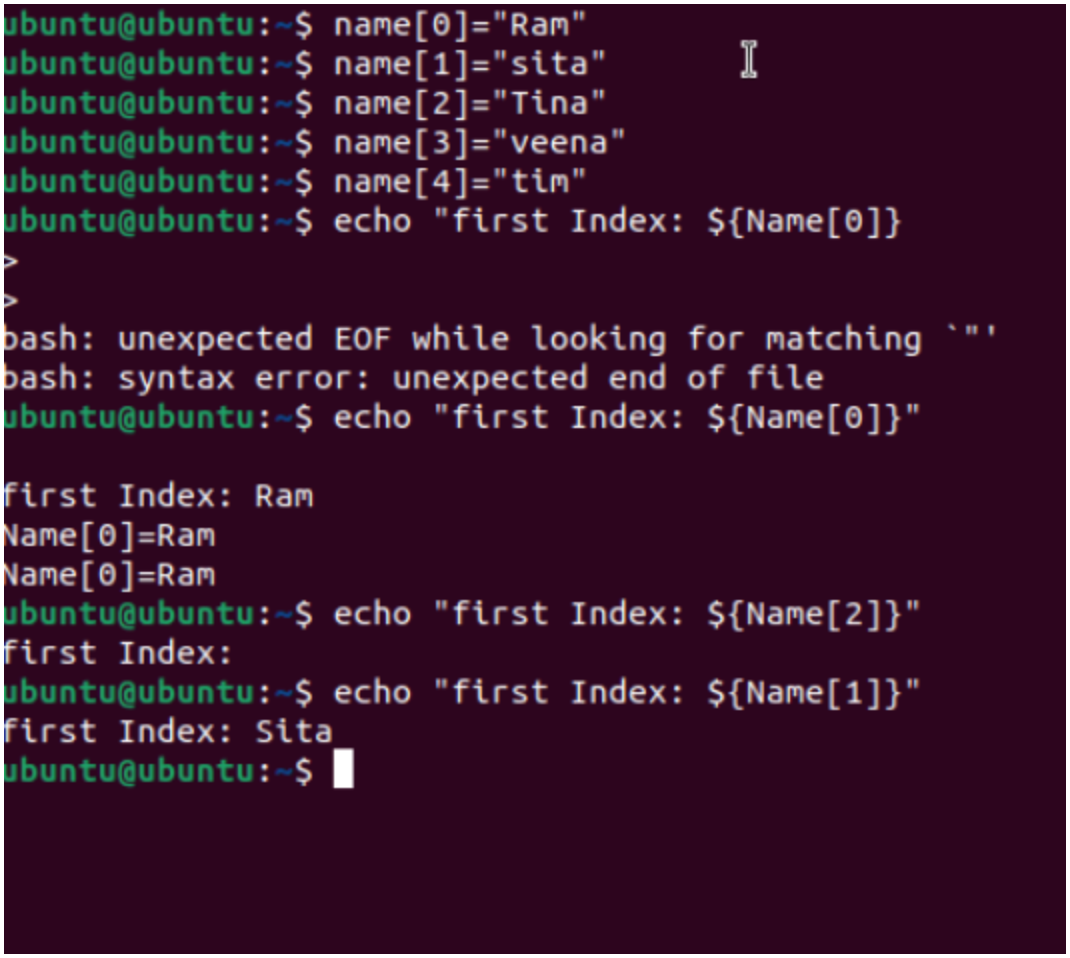


Task 33:

CAn u try to add a list of your friends names in an array and try to printout

Ex:

```
NAME[0]="Ram"

NAME[1]="Sita"

NAME[2]="Tina"

NAME[3]="Veena"

NAME[4]="Tim"

echo "First Index: ${NAME[0]}"

echo "Second Index: ${NAME[1]}"
```



Task 33:

CAn u try to add a list of your friends names in an array and try to printout

Ex:

NAME[0]="Ram"

NAME[1]="Sita"

NAME[2]="Tina"

NAME[3]="Veena"

NAME[4]="Tim"

echo "First Index: ${NAME[0]}"

echo "Second Index: ${NAME[1]}"

```
ubuntu@ubuntu:~$ name[0]="Ram"
ubuntu@ubuntu:~$ name[1]="sita"
ubuntu@ubuntu:~$ name[2]="Tina"
ubuntu@ubuntu:~$ name[3]="veena"
ubuntu@ubuntu:~$ name[4]="tim"
ubuntu@ubuntu:~$ echo "first Index: ${Name[0]}
>
>
bash: unexpected EOF while looking for matching `"'
bash: syntax error: unexpected end of file
ubuntu@ubuntu:~$ echo "first Index: ${Name[0]}"

first Index: Ram
Name[0]=Ram
Name[0]=Ram
ubuntu@ubuntu:~$ echo "first Index: ${Name[2]}"
first Index:
ubuntu@ubuntu:~$ echo "first Index: ${Name[1]}"
first Index: Sita
ubuntu@ubuntu:~$ friends=("Ram" "sita" "Tina" "veena"
```

Task 34:

Can you print all the list at once in an array.. Try the below cmds and check

Echo “${array_name[*]}”

Echo “${array_name[@]}”

Opetrators 👍

Arithmetic Operators

Relational Operators
Boolean Operators
String Operators
File Test Operators

If else

if...fi statement

if...else...fi statement

if...elif...else...fi statement

case...esac statement

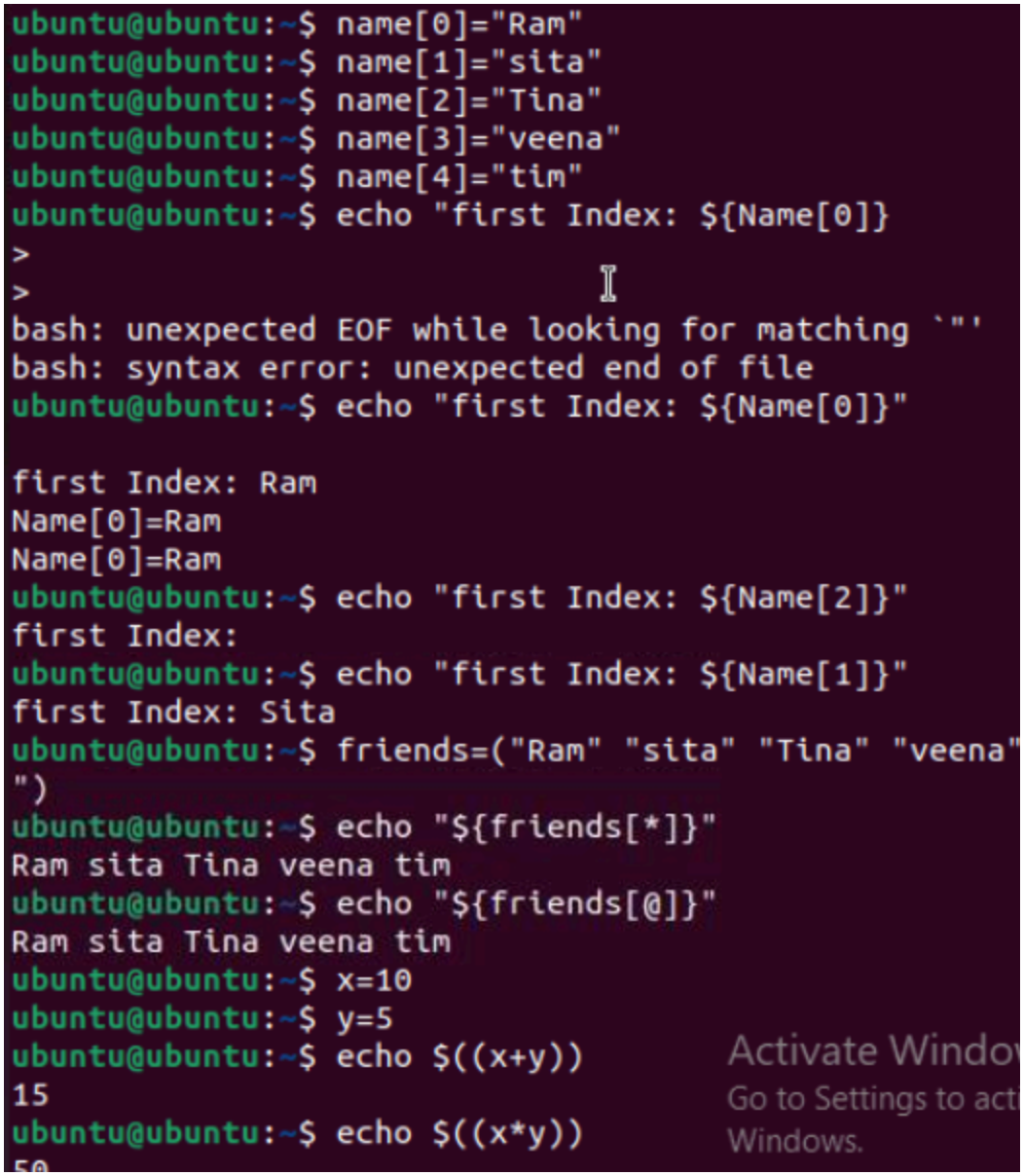
The while loop

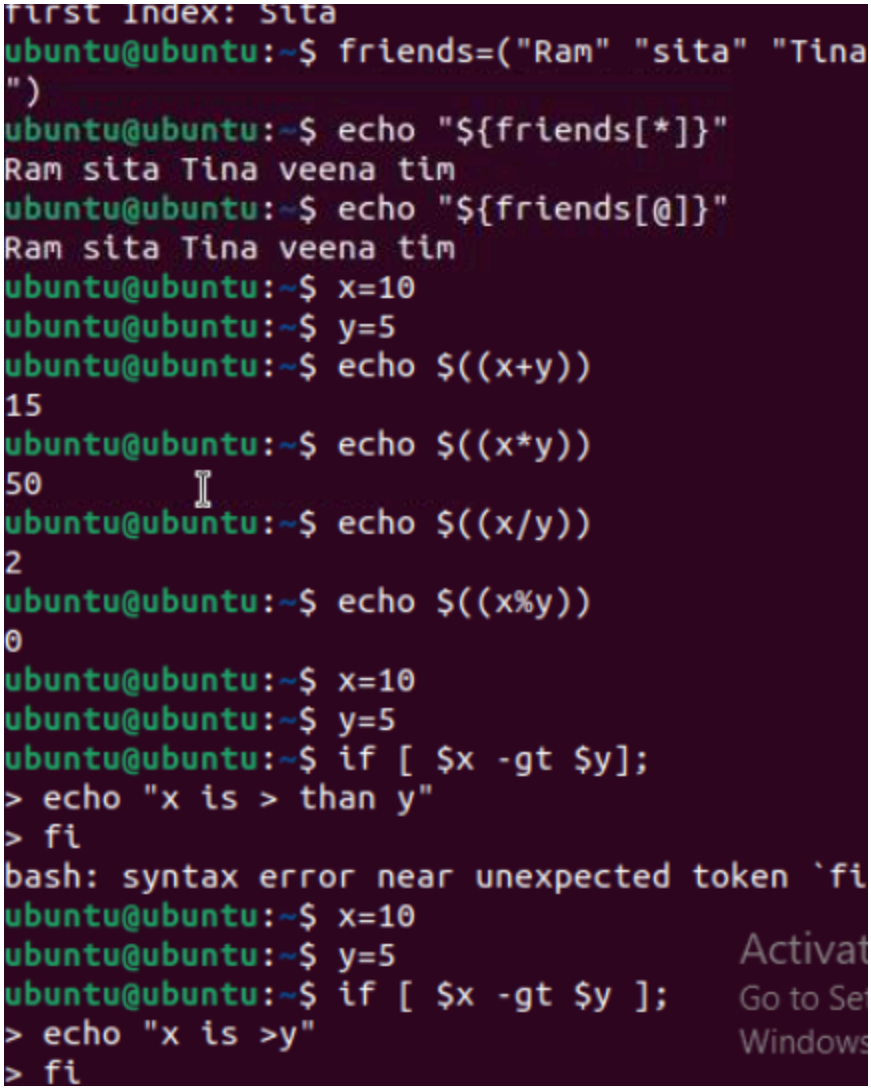
The for loop

The until loop

The select loop

Plz have an idea about the above


ubuntu@ubuntu:~$ name[0]="Ram"
ubuntu@ubuntu:~$ name[1]="sita"
ubuntu@ubuntu:~$ name[2]="Tina"
ubuntu@ubuntu:~$ name[3]="veena"
ubuntu@ubuntu:~$ name[4]="tim"
ubuntu@ubuntu:~$ echo "first Index: ${Name[0]}
>
>
bash: unexpected EOF while looking for matching `"'
bash: syntax error: unexpected end of file
ubuntu@ubuntu:~$ echo "first Index: ${Name[0]}"
first Index: Ram
Name[0]=Ram
Name[0]=Ram
ubuntu@ubuntu:~$ echo "first Index: ${Name[2]}"
first Index:
ubuntu@ubuntu:~$ echo "first Index: ${Name[1]}"
first Index: Sita
ubuntu@ubuntu:~$ friends=("Ram" "sita" "Tina" "veena"
")
ubuntu@ubuntu:~$ echo "${friends[*]}"
Ram sita Tina veena tim
ubuntu@ubuntu:~$ echo "${friends[@]}"
Ram sita Tina veena tim
ubuntu@ubuntu:~$ x=10
ubuntu@ubuntu:~$ y=5
ubuntu@ubuntu:~$ echo $((x+y))
15
ubuntu@ubuntu:~$ echo $((x*y))
50
Activate Windo
Go to Settings to act
Windows.



Task 35:

Plz let me know whats the output of the below snippet:

```
a=0

while [ "$a" -lt 10 ]    # this is loop1

do

  b="$a"

  while [ "$b" -ge 0 ]  # this is loop2
```

```
  do
    echo -n "$b "
    b=`expr $b - 1`
  done
  echo
  a=`expr $a + 1`
Done
```

```
ubuntu@ubuntu:~$ a=0
while [ "$a" -lt 10 ]; do
  b=$a
  while [ "$b" -ge 0 ]; do
    echo -n "$b "
    b=$((b - 1))
  done
  echo ""  # newline after inner loop
  a=$((a + 1))
done
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
ubuntu@ubuntu:~$
```