**day06\_107856406\_dsdipt\_sudipto\_29may2025**

**Employee Code:** 107856406

**Login ID:** dsdipt

**Email :** dsdipt@amazon.com

**Name:** Sudipto Das

**Date:** 29 May 2025 (Day 06)

### ***Task 1: RegEX Symbols in linux List them down with description***

**Regular Expression Symbols in Linux**

Regular expressions (regex) are powerful tools for pattern matching in text. In Linux, they are widely used in command-line utilities. Here's a breakdown of common symbols:

**Anchor Symbols**

**^ (Caret)**

* + **Description:** Matches the beginning of a line.
  + **Example:** ^start would match lines that begin with "start".

**$ (Dollar Sign)**

* + **Description:** Matches the end of a line.
  + **Example:** end$ would match lines that end with "end".

**\< or \b (Word Boundary - Beginning)**

* + **Description:** Matches the beginning of a word.
  + **Example:** \<word would match "word" only when it's at the beginning of a word (e.g., "wordplay" but not "unword").

**\> or \b (Word Boundary - End)**

* + **Description:** Matches the end of a word.
  + **Example:** word\> would match "word" only when it's at the end of a word (e.g., "keyword" but not "wordy").
  + **Note:** \b is a more portable and commonly used word boundary in many regex implementations, matching either the beginning or end of a word.

**Quantifier Symbols**

**\* (Asterisk)**

* + **Description:** Matches the preceding character zero or more times.
  + **Example:** a\*b would match "b", "ab", "aab", "aaab", etc.

**+ (Plus Sign)**

* + **Description:** Matches the preceding character one or more times.
  + **Example:** a+b would match "ab", "aab", "aaab", but not "b".

**? (Question Mark)**

* + **Description:** Matches the preceding character zero or one time (i.e., makes it optional).
  + **Example:** colou?r would match both "color" and "colour".

**{n} (Curly Braces - Exact Count)**

* + **Description:** Matches the preceding character exactly n times.
  + **Example:** a{3}b would match "aaab" only.

**{n,} (Curly Braces - Minimum Count)**

* + **Description:** Matches the preceding character at least n times.
  + **Example:** a{2,}b would match "aab", "aaab", "aaaab", etc.

**{n,m} (Curly Braces - Range Count)**

* + **Description:** Matches the preceding character at least n times but no more than m times.
  + **Example:** a{1,3}b would match "ab", "aab", "aaab".

**Character Class Symbols**

**. (Dot)**

* + **Description:** Matches any single character (except newline).
  + **Example:** a.b would match "axb", "ayb", "a1b", etc.

**[abc] (Square Brackets - Character Set)**

* + **Description:** Matches any one of the characters inside the brackets.
  + **Example:** [aeiou] would match any vowel.

**[^abc] (Square Brackets - Negated Character Set)**

* + **Description:** Matches any character *not* in the set.
  + **Example:** [^0-9] would match any non-digit character.

**[a-z] (Square Brackets - Range)**

* + **Description:** Matches any character within the specified range (e.g., lowercase letters).
  + **Example:** [A-Za-z0-9] would match any alphanumeric character.

**[:alnum:] (POSIX Character Class)**

* + **Description:** Matches alphanumeric characters (equivalent to [A-Za-z0-9]).

**[:alpha:] (POSIX Character Class)**

* + **Description:** Matches alphabetic characters (equivalent to [A-Za-z]).

**[:digit:] (POSIX Character Class)**

* + **Description:** Matches digits (equivalent to [0-9]).

**[:lower:] (POSIX Character Class)**

* + **Description:** Matches lowercase letters.

**[:upper:] (POSIX Character Class)**

* + **Description:** Matches uppercase letters.

**[:space:] (POSIX Character Class)**

* + **Description:** Matches whitespace characters (space, tab, newline, etc.).

**[:punct:] (POSIX Character Class)**

* + **Description:** Matches punctuation characters.

**Grouping and Alternation Symbols**

**| (Pipe / OR)**

* + **Description:** Acts as an OR operator, matching either the expression before or after it.
  + **Example:** cat|dog would match "cat" or "dog".

**() (Parentheses - Grouping)**

* + **Description:** Groups parts of the regular expression, allowing quantifiers to apply to the entire group or for backreferencing.
  + **Example:** (ab)+ would match "ab", "abab", "ababab", etc.

**Escape Character**

**\ (Backslash)**

* + **Description:** Escapes the special meaning of a regex symbol, treating it as a literal character.
  + **Example:** \$ would match a literal dollar sign, not the end-of-line anchor. \. would match a literal dot, not any character.

### ***Task 2: What are the important features of Linux OS?***

* **Open Source & Free:** Freely available source code, no licensing costs, driven by a global community.
* **Stability & Reliability:** Known for high uptime and robust performance, ideal for critical systems.
* **Security:** Strong built-in security features, user permissions, and rapid community vulnerability patching.
* **Multi-user & Multitasking:** Supports multiple users working concurrently and efficiently runs many tasks simultaneously.
* **Powerful CLI & Scripting:** Offers a highly efficient command-line interface for granular control and automation.

### ***Task 3: Explain Kernel & its functions.***

**What is the Kernel?**

The **Kernel** is the **core** or **heart** of an operating system (OS). It's the first program loaded when a computer starts up, and it remains in memory until the computer is shut down. Think of it as the **central control unit** or **manager** of our computer's hardware resources.

**Kernel Function**

* **Process Management:** Manages when and how programs run, ensuring multiple applications can share the CPU.
  + **Example:** Decides whether your web browser or music player gets CPU time next.
* **Memory Management:** Allocates and protects memory space for all running programs to prevent conflicts.
  + **Example:** Ensures your document editor doesn't accidentally write over your spreadsheet's data.
* **Device Management:** Controls all hardware components, allowing software to interact with them via drivers.
  + **Example:** Translates your "print" command into actions for the printer.
* **System Call Interface:** Provides a standardized way for applications to request services from the kernel.
  + **Example:** When an app needs to save a file, it uses a specific "write" system call.
* **File System Management:** Organizes and handles how files are stored, retrieved, and managed on storage devices.
  + **Example:** Locates and opens your presentation file on the hard drive when you click on it.

### ***Task 4: What is BASH? Full form with explanation.***

**BASH** stands for **B**ourne **A**gain **SH**ell.

**Bourne Again:**

**"Bourne"** refers to the **Bourne Shell (sh)**, which was one of the earliest and most influential Unix shells, written by Stephen Bourne.

**"Again"** signifies that BASH is a **successor and enhancement** to the original Bourne Shell. It was developed by Brian Fox for the GNU Project as a free software replacement for the Bourne Shell.

**Shell:**

In Linux (and other Unix-like operating systems), a "shell" is a **command-line interpreter**. It's a program that provides a command-line interface (CLI) for users to interact with the operating system. When we open a terminal window, we are typically interacting with a shell. It takes our typed commands, interprets them, and then executes them, often by telling the kernel what to do.

### ***Task 5: What is the difference between windows and linux.***

**Licensing & Source Code:** **Windows is proprietary and closed-source**, requiring a paid license. **Linux is open-source and generally free**, allowing anyone to view, modify, and distribute its code.

**Security & Stability:** **Linux is often considered more secure and exceptionally stable**, rarely requiring reboots and highly resistant to common malware. **Windows, being a larger target, faces more malware threats** and often necessitates more frequent reboots for updates.

**User Interface & Control:** **Windows primarily offers a consistent graphical user interface (GUI)**. **Linux provides a wide variety of customizable GUIs** (desktop environments) alongside a powerful and deeply integrated command-line interface (CLI) for advanced control.

**Software Ecosystem:** **Windows has native support for a vast range of commercial software and games** (e.g., Microsoft Office, Adobe Creative Suite). **Linux thrives on a rich open-source software ecosystem**, though it sometimes requires alternative applications for specific proprietary software.

**Target Audience:** **Windows is the dominant choice for general desktop users, gamers, and businesses** relying on specific commercial software. **Linux is preferred by developers, system administrators, and server environments**, as well as users who prioritize customization, control, and efficiency.

### ***Task 7: Is it legal to edit Kernel?***

Yes, it is **absolutely legal** to edit the Linux Kernel.

This is because the Linux kernel is **open-source** and licensed under the **GNU General Public License (GPLv2)**. This license specifically grants users the freedom to:

* **Study** how the program works.
* **Modify** the program as they wish.
* **Distribute** copies of the original or modified versions, provided the modified source code is also made available under the same license.

### ***Task 8: Can you explain LILO***

**LILO (LInux LOader)** was an early and popular **bootloader** for Linux.

Its main function was to **load the Linux kernel** into memory, thereby starting the operating system, after the computer's BIOS finished its initial checks. LILO worked by directly mapping the physical location of the kernel on the disk and required users to manually run the lilo command to update its configuration whenever the kernel or boot setup changed.

It has largely been **replaced by GRUB (GRand Unified Bootloader)**, which is more advanced, user-friendly, and dynamically configured, making LILO a legacy bootloader today.

### ***Task 9: What is a shell? How many shells are there? Explain.***

**What is a Shell?**

In Linux (and other Unix-like operating systems), a **shell** is a **command-line interpreter**. It's essentially a program that acts as an **interface between the user and the operating system's kernel**.

* **translator:** You type commands in plain text (e.g., ls -l, cd /home/user), and the shell translates those commands into instructions that the kernel can understand and execute.
* **command prompt:** When you open a terminal window, you're interacting with a shell. It provides the prompt where you type your commands.
* **An environment for scripting:** Shells are also powerful programming environments that allow you to write "shell scripts" – sequences of commands to automate tasks.

**Here are the 5 most popular shells with a short description and key feature:**

1. **Bash (Bourne-Again SHell)**
   * **Short:** The most common and default Linux shell.
   * **Key Feature:** Versatile for both interactive use and robust scripting.
2. **Zsh (Z Shell)**
   * **Short:** A modern, highly customizable shell.
   * **Key Feature:** Advanced auto-completion, powerful themes/plugins (e.g., Oh My Zsh).
3. **Sh (Bourne Shell)**
   * **Short:** The original Unix shell.
   * **Key Feature:** Provides fundamental syntax for highly portable scripts.
4. **Ksh (Korn Shell)**
   * **Short:** A powerful shell blending Bourne and C shell features.
   * **Key Feature:** Strong capabilities for enterprise-level scripting.
5. **Fish (Friendly Interactive SHell)**
   * **Short:** Designed for user-friendliness.
   * **Key Feature:** Excellent out-of-the-box auto-suggestions and syntax highlighting.

### ***Task 10: What is Swap space?***

**Swap space** is a designated area on a hard disk drive (or SSD) that a Linux operating system uses as a **temporary extension of its physical memory (RAM)**.

It essentially **converts a portion of your hard disk into virtual memory**, allowing your system to move less-used data from RAM to disk when physical RAM is full. This prevents applications from crashing and helps the system remain stable, especially when running many programs or memory-intensive tasks.

### 

### ***Task 11: What is Mount? How do you mount and unmount a file in Linux?***

**What is Mount?**

**Mounting** in Linux is the process of **making a storage device (like a USB drive, hard drive partition, or CD-ROM) accessible** to your operating system. It involves connecting the device's file system to a specific, empty directory (called a **mount point**) within your existing Linux file system tree, making its contents visible and usable.

**How to Mount and Unmount**

**Mount a File System:**

**Command:** sudo mount /dev/sdXN /mnt/mydata

* + sudo: Runs the command with administrative privileges.
  + /dev/sdXN: The device name (e.g., /dev/sdb1 for a USB partition).
  + /mnt/mydata: The existing (empty) directory where you want to access the device's contents (your **mount point**).

**Unmount a File System:**

**Command:** sudo umount /mnt/mydata (using the mount point) OR sudo umount /dev/sdXN (using the device name)

### ***Task 12: What is the chmod command ? how to use it?***

chmod (change mode) is a fundamental Linux command used to **change file and directory permissions**.

It controls who can **read (r), write (w), or execute (x)** files or directories for the **owner**, **group**, and **others**.

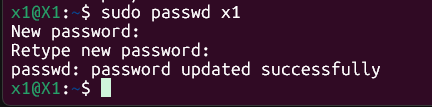
1. **Symbolic Mode:** Use letters (u=owner, g=group, o=others, a=all) with + (add), - (remove), or = (set) and permission types (r, w, x).
   * **Example:** chmod u+x script.sh (gives owner execute permission).
   * **Example:** chmod go-w file.txt (removes write permission for group and others).
2. **Numeric (Octal) Mode:** Use a 3-digit number where each digit represents permissions for owner, group, and others. Each permission has a value: r=4, w=2, x=1. Sum the values for each category.
   * **Example:** chmod 755 script.sh (owner: rwx=7; group: r-x=5; others: r-x=5).
   * **Example:** chmod 644 file.txt (owner: rw-=6; group: r--=4; others: r--=4).

### ***Task 13: Can we add a new user account? Create a new user in different ways.***

**We can create a new user in Linux using two primary commands:**

1. **sudo adduser [username]**:
   * **User-friendly:** Prompts for password, automatically creates home directory and sets up default configurations. Recommended for most interactive user creation.
2. **sudo useradd -m -s /bin/bash [username]** followed by **sudo passwd [username]**:
   * **Low-level:** Provides more granular control, requires explicit options (like -m for home directory) and a separate command (passwd) to set the password. Used for scripting or specific needs.

### ***Task 14: Can you change the password of a user, How to do that?***



### ***Task 15: What is the difference between Process and Thread?***

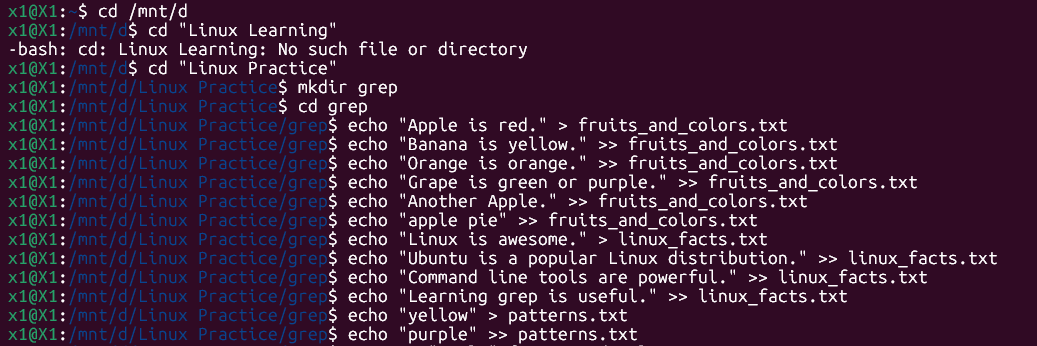
**Process:**

* An **independent, self-contained running program.**
* Has its **own isolated memory space and resources.**
* Think of it as a separate application running (e.g., your web browser or a word processor).
* **Heavyweight:** More resource-intensive to create and switch between.

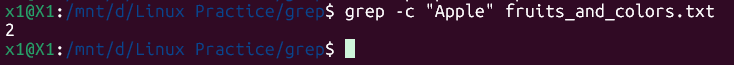
**Thread:**

* A **single unit of execution *within* a process.**
* Multiple threads can run concurrently within the **same process**, sharing its memory space and resources.
* Think of different tasks being performed simultaneously *within* one application (e.g., in your web browser, one thread renders the page while another downloads an image).
* **Lightweight:** Less resource-intensive to create and switch between than processes.

### ***Task 16: Practice grep***

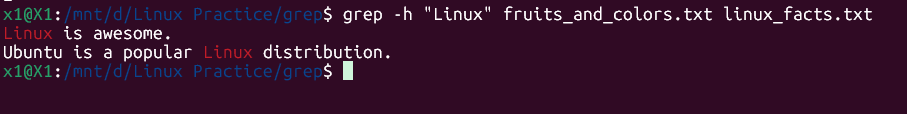


**1. grep -c (Count of matching lines)**



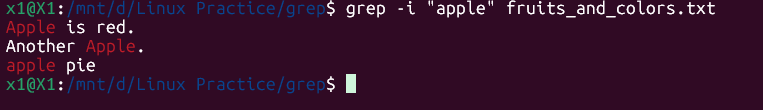
**Shows 2 (because "Apple" appears on 2 lines).**

**2. grep -h (Display matched lines, but no filenames)**



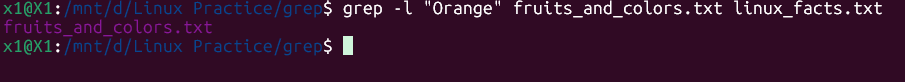
**Shows lines directly, like "Linux is awesome."**

**3. grep -i (Ignores case for matching)**



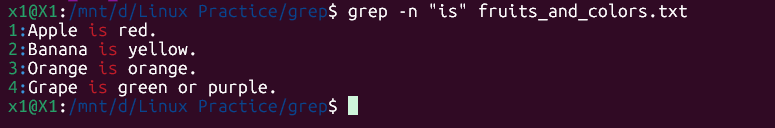
**Matches both "Apple" and "apple".**

**4.grep -l (Displays list of filenames only)**



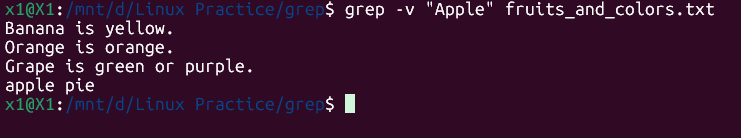
**Shows fruits\_and\_colors.txt.**

**5. grep -n (Display matched lines and their line numbers)**



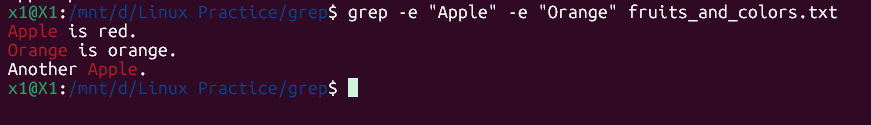
**Shows 1:Apple is red., 2:Banana is yellow., etc.**

**6. grep -v (Print lines that do NOT match the pattern)**



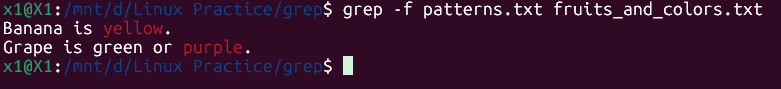
**Shows all lines *except* those containing "Apple".**

**7. grep -e exp (Specifies expression with this option. Can use multiple times.)**



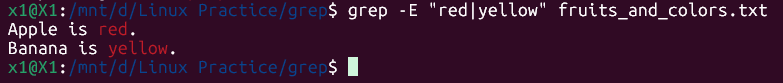
**Matches lines with "Apple" *or* "Orange".**

**8. grep -f file (Takes patterns from file, one per line.)**



**Searches for patterns listed in patterns.txt (like "yellow" or "purple").**

**9. grep -E (Treats pattern as an extended regular expression (ERE))**



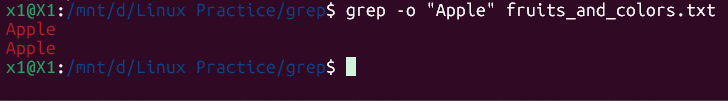
**Matches lines containing "red" or "yellow" using ERE.**

**10. grep -w (Match whole word)**



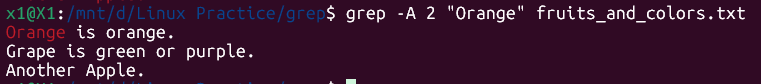
**Matches "apple" but not "Apple" or "Another Apple."**

**11. grep -o (Print only the matched parts)**



**Shows Apple on one line, then Apple on another.**

**12. grep -A n (After Context)**



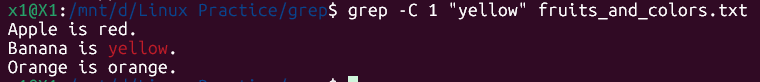
**It shows the line with "Orange" and the next 2 lines**

**13. grep -B n (Before Context)**

****

**It shows the line with "Another Apple." and the 1 line preceding it**

**14. grep -C n (Context - Before and After)**

****

**It shows the line with "yellow", the 1 line before it, and the 1 line after it**

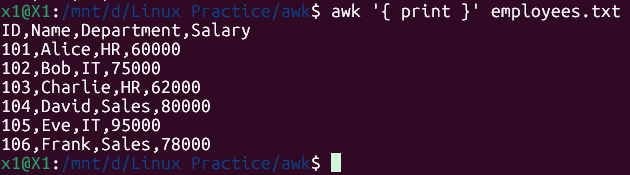
### ***Task 17: Practice awk Linux command.***

awk is a **powerful text processing tool and a programming language** that processes files line by line.

It's used for:

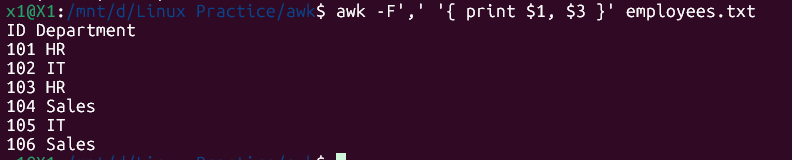
* **Pattern matching:** Finding lines that meet specific criteria.
* **Data extraction:** Pulling out specific columns (fields) of information.
* **Text reformatting:** Changing the layout or structure of text.
* **Generating reports:** Summarizing and presenting data.

**Example 1: Print Contents of a File**



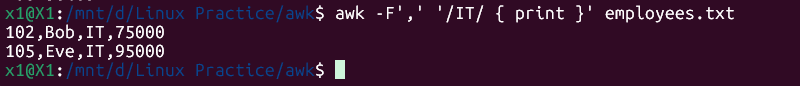
**Prints every line of the employees.txt file to the console.**

**Example 2: Print Specific Columns of a File**



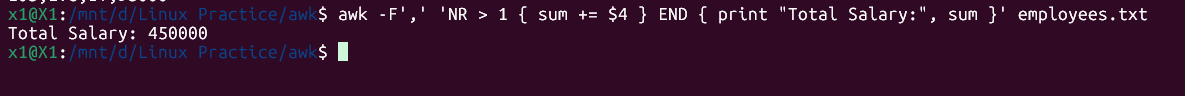
**Prints the first ($1) and third ($3) columns of each line.**

**Example 3: Filter Lines Based on a Condition**



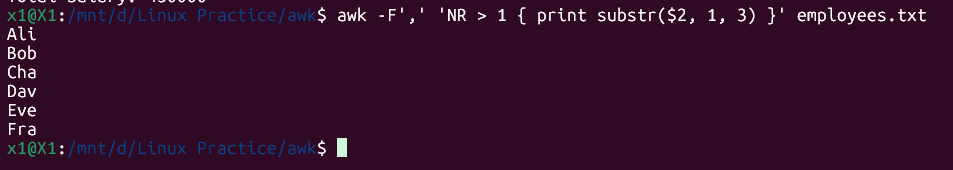
**Prints only lines that contain the word "IT"**

**Example 4: Sum Values in a Column**



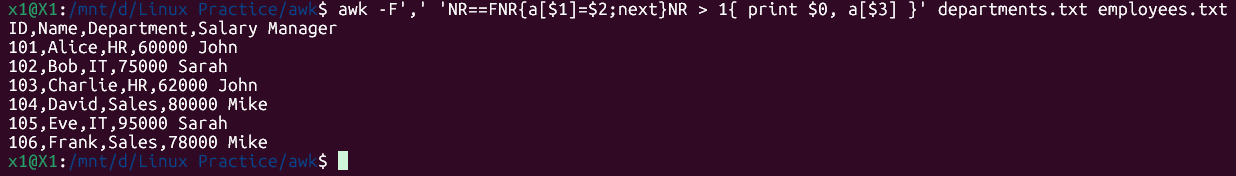
**Calculates and prints the total sum of all values in the fourth column ($4, which is Salary).**

**Example 5: Extract Substring from a Column**



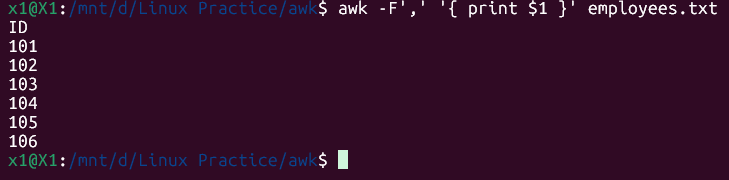
**Extracts and prints the first three characters (starting from position 1, for a length of 3) from the second column ($2) of each line.**

**Example 6: Join Two Files Based on a Common Column**



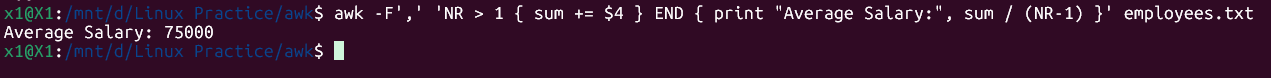
**Joins employees.txt and departments.txt based on the Department column ($3 in employees, $1 in departments), appending the manager's name from departments.txt.**

**Example 7: Extract First Column of a File**



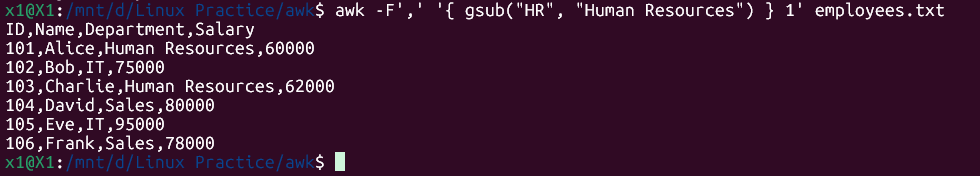
**Prints only the first column ($1) of each line.**

**Example 8: Calculate Average of a Column**



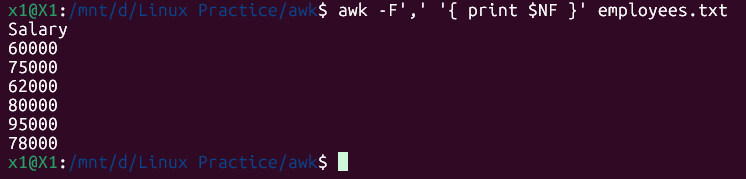
**Calculates the average of values in the fourth column ($4, Salary) by summing them and dividing by the total number of data records (NR-1).**

**Example 9: Replace a String in a File**

****

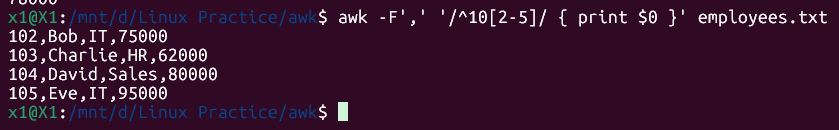
**Replaces all occurrences of "HR" with "Human Resources" within the file.**

**Example 10: Display Last Field of a File**



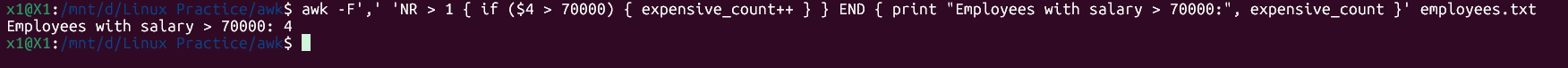
**Prints the last field ($NF) of each line.**

**Example 11: Using Regular Expressions**



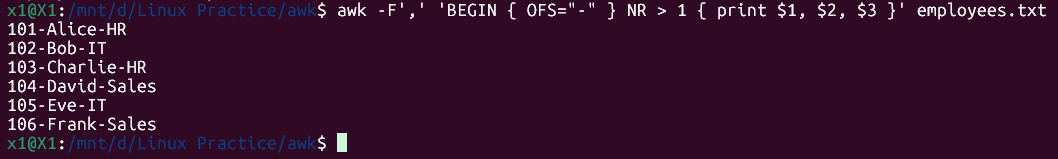
**Prints any line that contains the specified pattern (^10[2-5], meaning lines starting with "10" followed by 2, 3, 4, or 5) using a regular expression.**

**Example 12: Using Variables**



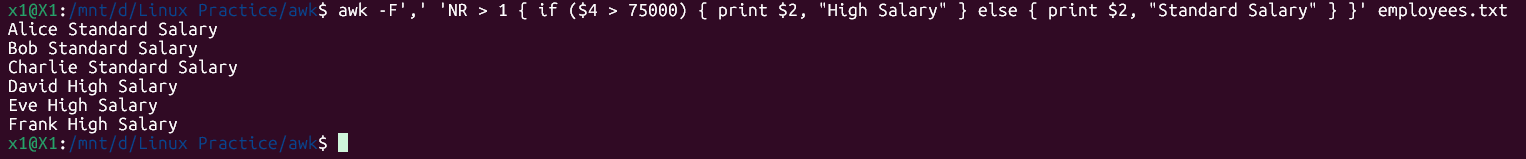
**Uses a variable (expensive\_count) to count employees whose salary (4th column) is over 70000, and prints the final count.**

**Example 13: Using Built-in Variables**



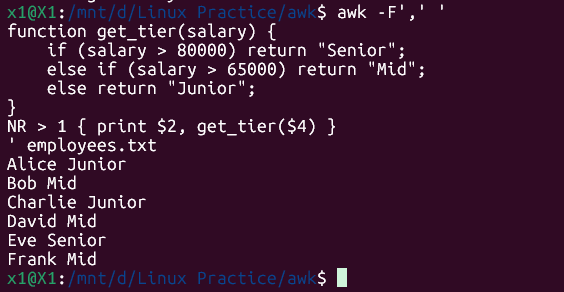
**Sets the Output Field Separator (OFS) to a hyphen -, so printed fields are separated by hyphens instead of spaces.**

**Example 14: Using Control Statements (if-else)**



**Performs conditional operations; if the salary ($4) is greater than 75000, it prints "High Salary" for the employee, otherwise "Standard Salary".**

**Example 15: Using Functions**



**Defines and uses a custom function (get\_tier) to categorize salaries into "Junior", "Mid", or "Senior" tiers.**

**AWK Keywords:**

* **NR (Number of Record)**
* **FNR (File Number of Record)**
* **-F (Field Separator Command-line Option)**
* **FS (Field Separator Variable)**
* **RS (Record Separator Variable)**
* **OFS (Output Field Separator Variable)**
* **ORS (Output Record Separator Variable)**
* **$NF is a special built-in variable in AWK.dynamically points to the value of the last field,**

### ***Task 18: How to check file access permission in Linux?***

To check file access permissions in Linux, the most common and effective command is ls with the -l (long listing) option.

**Using ls -l**

### ***Task 19: What are the default permissions for a new file?***

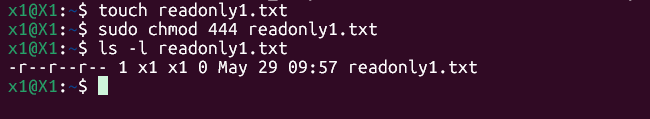
**When a new file is created, the default permissions are:**

**rw-r--r--**

That corresponds to:

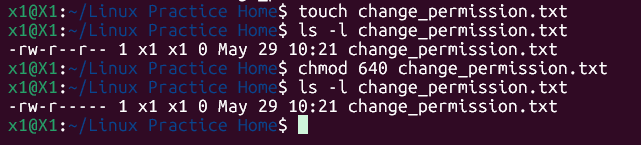
* **Owner**: Read + Write (rw-)
* **Group**: Read-only (r--)
* **Others**: Read-only (r--)

### ***Task 20: command to change the permission to read only for everyone.***



### ***Task 21: Can you change the file permissions to match the following:***

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

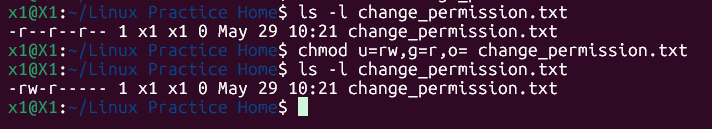
****

**Explanation of 640:**

* 6 (Owner): Represents Read (4) + Write (2).
* 4 (Group): Represents Read (4).
* 0 (Others): Represents No permissions (0).

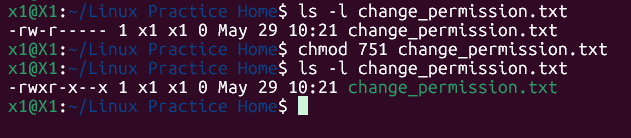
### ***Task 22: What was the command for changing the file permissions to -rw-r-----?***

**chmod u=rw,g=r,o= filename**

****

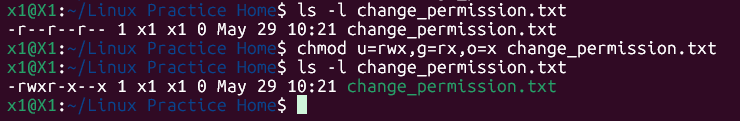
### ***Task 23: Change chmod.exercises permissions to -rwxr-x--x***

* **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***

**chmod u=rwx,g=rx,o=x filename**

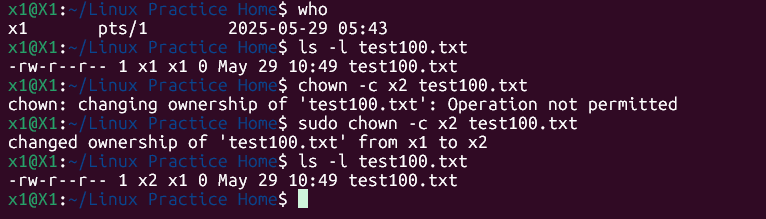


### ***Task 25: what will the below command do?***

chown -c master file1.txt

The command chown -c master file1.txt will change the owner of the file named file1.txt to the user master.

* **chown:** This is a Linux command that stands for "change owner". It is used to change the user owner and/or group owner of a given file, directory, or symbolic link.
* **-c:** This is an option that stands for "changes made". It tells chown to be verbose and report only when a change is made. If the ownership of file1.txt is already master, it will not print anything. If it successfully changes the owner, it will print a message confirming the change.
* **master:** This is the new username that will become the owner of the file. This user master must exist on the system.
* **file1.txt:** This is the target file whose ownership will be changed.



### ***Task 26: Define what is a process***

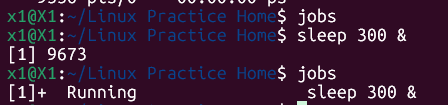
**A process in Linux is an instance of a running program, along with all the resources it uses (like memory and CPU time). Each time you run a command or an application, a process is created.**

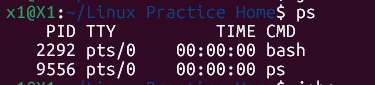
### ***Task 27: command to check foreground & background process***

* A **foreground process** is active in your terminal, taking input and displaying output.
* A **background process** runs independently, allowing you to use the terminal for other commands.

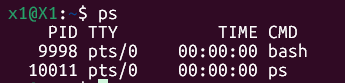
**To check them:**

* **jobs**: Lists background/stopped processes in your current shell.
* **ps:** Lists all running processes on the system





### ***Task 28: list all the running processes***



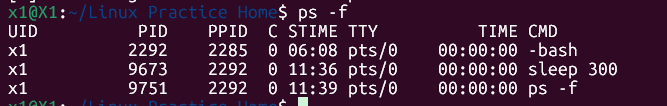
### ***Task 29: What will ps -f command do?***

The ps -f command in Linux is used to display a **full-format listing** of your own processes.

**What ps -f Does:**

The -f option (full format) provides more detailed information about processes than the default ps command. It's particularly useful for seeing hierarchical relationships between processes and more context about their execution.

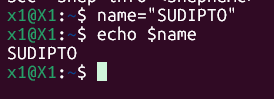
When we run ps -f, it typically displays the following columns:



* **UID**: User ID (the user who owns the process).
* **PID**: Process ID (a unique identification number for the process).
* **PPID**: Parent Process ID (the ID of the process that started this process).
* **C**: CPU utilization (an estimate of CPU usage).
* **STIME**: Start Time (the time when the process started).
* **TTY**: Controlling Terminal (the terminal associated with the process).
* **TIME**: Cumulative CPU time (total CPU time consumed by the process).
* **CMD**: Command (the command that started the process, often with its arguments).

### 

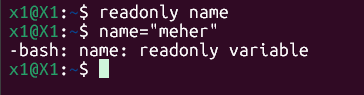
### ***Task 30: create a variable name with your name (Ex: Name = “SUDIPTO”)***



**In this example:**

* name is the variable name.
* = is the assignment operator.
* "SUDIPTO" is the value assigned to the variable.

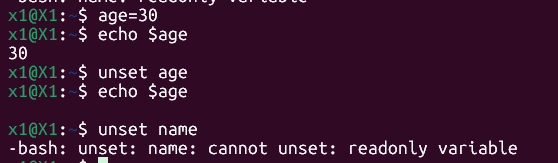
### ***Task 31: make the name variable read only.***



**This message confirms that the name variable is read-only and its value cannot be changed to "meher". It will retain its original value.**

### ***Task 32: unset or delete the variable***

**To delete variables in a Linux shell, we use the unset command.**



We **cannot unset a variable that has been made readonly**.

When a shell is running, there are indeed three main types of variables present:

**Local Variables:**

* + **Definition:** These variables are created and exist only within the current shell session or the specific script in which they are defined. They are not passed on to child processes (subshells).
  + **Creation:** They are typically created by simply assigning a value, like my\_var="hello".

**Environment Variables:**

* + **Definition:** These variables are made available to all processes that are spawned from the current shell (child processes). They are often used to define system-wide or user-specific settings that affect how programs behave.
  + **Creation:** They are usually created by assigning a value and then using the export command, like export PATH="/usr/local/bin:$PATH". Common examples include PATH, HOME, USER, and SHELL.

**Shell Variables (or Special Variables):**

* + **Definition:** These are special variables set and maintained by the shell itself. They typically hold information about the shell's state, the last command executed, arguments passed to a script, or current process IDs.
  + **Examples:**
    - $0: The name of the shell script being executed.
    - $#: The number of arguments passed to a script.
    - $\* or $@: All arguments passed to a script.
    - $?: The exit status of the most recently executed foreground command.
    - $$: The process ID (PID) of the current shell.
    - $!: The PID of the last background command.

### ***Task 33: try to add a list of your friends names in an array and try to print out***

Example:

NAME[0]="Ram"

NAME[1]="Sita"

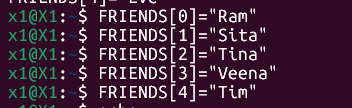
NAME[2]="Tina"

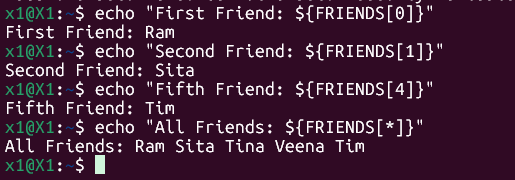
NAME[3]="Veena"

NAME[4]="Tim"

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

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

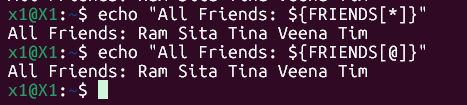




### ***Task 34: print all the list at once in an array.***

**Echo “${array\_name[\*]}”**

**Echo “${array\_name[@]}”**



### ***Task 35: what's 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 "**

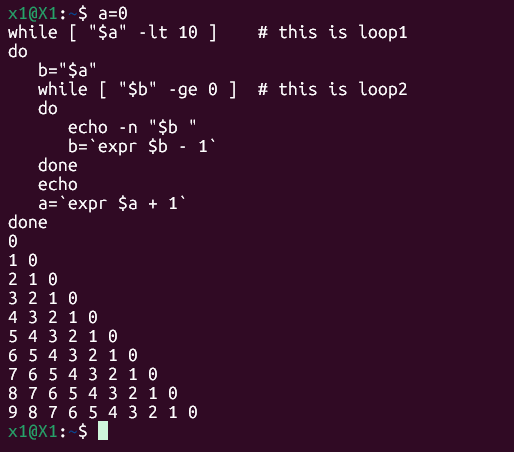
**b=`expr $b - 1`**

**done**

**echo**

**a=`expr $a + 1`**

**Done**



**Snippet Analysis:**

**Outer Loop (a):**

* + a=0: Initializes a variable a to 0.
  + while [ "$a" -lt 10 ]: This loop will continue as long as the value of a is less than 10. So, a will take values from 0 up to 9.
  + a=expr $a + 1``: Increments a by 1 at the end of each iteration of the outer loop.

**Inner Loop (b):**

* + b="$a": At the beginning of each outer loop iteration, a new variable b is initialized with the current value of a.
  + while [ "$b" -ge 0 ]: This loop will continue as long as the value of b is greater than or equal to 0.
  + echo -n "$b ": This is crucial! It prints the current value of b followed by a space, and the -n option prevents it from adding a newline character. This means numbers will be printed on the same line.
  + b=expr $b - 1``: Decrements b by 1 in each iteration of the inner loop.

**echo (between loops):** After the inner loop finishes (meaning b has counted down to 0 and gone below it), an echo command without any arguments is executed. This prints a newline character, moving the cursor to the beginning of the next line for the output of the next outer loop iteration.