Experiment 2(editors like vi,vim and nano)

| **Task** | **vi/vim** | **nano** |
| --- | --- | --- |
| Open File | vi filename | nano filename |
| Save File | :w | Ctrl + O |
| Exit | :q | Ctrl + X |
| Save & Exit | :wq | Ctrl + O, then Ctrl + X |
| Force Quit | :q! | Ctrl + X, then N |
| Insert Text | i, a, o | Type directly |
| Delete Line | dd | Ctrl + K |
| Undo | u | Not available (basic nano) |

**With nano (easiest):**

1. Open terminal.
2. Type:

bash

CopyEdit

nano myfile.txt

1. Start typing!
2. To save: Press Ctrl + O → Enter
3. To exit: Press Ctrl + X

**With vim or vi (a bit advanced):**

1. Open terminal.
2. Type:

bash

CopyEdit

vi myfile.txt

1. You’ll see the file opened in **command mode**.
2. Press i to switch to **insert mode** (to start typing).
3. After typing, press Esc to go back to command mode.
4. To save and quit:

bash

CopyEdit

:wq

(Type colon :, then wq, then press Enter)

**🧠 What is vi, vim, and nano?**

**🔹 vi (Visual Editor)**

* vi is one of the **oldest and most standard** text editors in Unix/Linux systems.
* It runs in the terminal and works in different **modes** (like command mode and insert mode).
* It's **lightweight** and always available — even in very basic systems.

**🔹 vim (Vi IMproved)**

* vim is an upgraded version of vi with **more features**, like syntax highlighting, undo/redo, and more.
* It’s more powerful and often used by programmers and sysadmins.

**🔹 nano**

* nano is a **very beginner-friendly editor**.
* It’s **easier to use** than vi or vim because it shows the commands (like Save, Exit) at the bottom of the screen.
* You can just open the file and start typing — no modes like in vi.

**✅ Why are they used?**

These editors are used to:

* **Create and edit files** (like scripts, configuration files, code).
* Work directly inside the **terminal or command line**.
* Modify system files, logs, or write code when you don’t have access to GUI editors like VS Code.

**Experiment 3**

Execution of Unix General Purpose Utility Commands like echo, clear, exit,  
date, time, uptime, cal, cat, tty, man, which, history, id, pwd, whoami, ping,  
ifconfig, pr, lp, lpr, lpstat, lpq, lprm, cancel, mail, etc.

| **Command** | **What it Does** | **Example Usage** |
| --- | --- | --- |
| echo | Prints text to the terminal | echo "Hello World" |
| clear | Clears the terminal screen | clear |
| exit | Exits the terminal or script | exit |
| date | Displays current date and time | date |
| time | Shows the time taken by a command | time ls |
| uptime | Shows how long the system has been running | uptime |
| cal | Displays a calendar | cal |
| cat | Displays file content | cat filename.txt |
| tty | Displays the terminal you're using | tty |
| man | Opens manual for a command | man ls |
| which | Shows the path of a command | which python3 |
| history | Shows list of recently used commands | history |
| id | Displays user ID and group ID | id |
| pwd | Shows current working directory | pwd |
| whoami | Shows the current logged-in user | whoami |
| ping | Checks network connectivity | ping google.com |
| ifconfig | Shows IP/network interface info (use ip a on modern systems) | ifconfig |
| pr | Formats file for printing | pr filename.txt |
| lp | Sends file to default printer | lp file.txt |
| lpr | Another print command | lpr file.txt |
| lpstat | Shows print queue status | lpstat |
| lpq | Displays the printer queue | lpq |
| lprm | Removes a job from print queue | lprm 123 |
| cancel | Cancels a print job | cancel job-id |
| mail | Opens mail utility (if configured) | mail |

**Experiment 4**

Execution of File System Management Commands like ls,, rmdir, rm, cp, mv, chmod, wc, piping and redirection, grep, sort, head, tail, diff, comm, less, more, file, type, wc, split, cmp, tar, find,  
vim, gzip, bzip2, unzip, locate, etc.

### 🔧 ****Basic File System Commands****

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| ls | Lists files in a directory | ls -l |
| rm | Removes a file | rm file.txt |
| rmdir | Removes an empty directory | rmdir folder/ |
| cp | Copies files or directories | cp file1.txt file2.txt |
| mv | Moves or renames files | mv oldname.txt newname.txt |
| chmod | Changes file permissions | chmod 755 script.sh |
| wc | Word, line, byte count | wc filename.txt |

### 🧪 ****Text Processing and Utilities****

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| grep | Searches text using patterns | grep "hello" file.txt |
| sort | Sorts lines in a file | sort names.txt |
| head | Shows first lines | head -n 10 file.txt |
| tail | Shows last lines | tail -n 5 file.txt |
| diff | Compares two files line-by-line | diff file1.txt file2.txt |
| comm | Finds common lines | comm file1.txt file2.txt |
| cmp | Compares two files byte-by-byte | cmp file1.bin file2.bin |
| wc | Counts lines/words/characters | wc -l file.txt |
| split | Splits files into pieces | split -l 100 file.txt |

### 🔁 ****Piping and Redirection****

| **Symbol** | **Function** | **Example** |
| --- | --- | --- |
| > | Redirects output to a file (overwrite) | ls > list.txt |
| >> | Appends output to file | echo "Hi" >> log.txt |
| < | Takes input from a file | wc < file.txt |
| ` | ` | Piping (output of one cmd to another) |

### 🗂️ ****Viewing Files****

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| more / less | View file page by page | less longfile.txt |
| file | Detects file type | file image.jpg |
| type | Shows command type | type ls |

### 📦 ****Compression & Archive****

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| tar | Archives files | tar -cvf archive.tar file1 file2 |
| gzip | Compresses files | gzip file.txt |
| bzip2 | Compresses files (better) | bzip2 file.txt |
| unzip | Extracts zip archives | unzip archive.zip |

### 🔍 ****Search Utilities****

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| find | Finds files with conditions | find . -name "\*.txt" |
| locate | Quickly locates files (needs indexing) | locate file.txt |

**Experiment 5**

Execution of User Management Commands like who, whoami, su, sudo, login,logout, exit, useradd/adduser, usermod, userdel, groupadd, groupmod,groupdel, gpasswd, chown, chage, chgrp, chfn, etc.

**👤 User Identification and Session**

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| who | Shows who is logged in | who |
| whoami | Shows your current username | whoami |
| login | Logs into a system (used in terminals) | login (used during boot or on TTY) |
| logout | Logs out of the current session | logout (used in shell) |
| exit | Closes the shell/terminal | exit |

**🔐 Switching Users and Permissions**

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| su | Switch user | su - username |
| sudo | Run command as superuser | sudo apt update |

**👥 User Management**

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| useradd / adduser | Adds a new user | sudo useradd john or sudo adduser john |
| usermod | Modifies user account | sudo usermod -aG sudo john (adds john to sudo group) |
| userdel | Deletes a user | sudo userdel john |

**👨‍👩‍👧‍👦 Group Management**

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| groupadd | Adds a new group | sudo groupadd developers |
| groupmod | Modifies group name or ID | sudo groupmod -n devs developers |
| groupdel | Deletes a group | sudo groupdel devs |
| gpasswd | Administers group passwords | sudo gpasswd -a john devs (adds user to group) |

**🛡️ Ownership & Access Control**

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| chown | Changes file owner | sudo chown john file.txt |
| chgrp | Changes group ownership | sudo chgrp devs file.txt |
| chage | Sets password expiration | sudo chage -l john (shows password aging info) |
| chfn | Changes user info (like full name) | sudo chfn john |

* Use man <command> to learn more about any command (e.g., man useradd).
* Try on a **Linux VM** or **WSL (Windows Subsystem for Linux)** or any **online Linux terminal**.

**Experiment 6**

a. Execution of Process Management Commands like ps, pstree, nice, kill, pkill, killall, xkill, fg, bg, pgrep, renice, etc  
b. Execution of Memory Management Commands like free, /proc/meminfo, top, htop, df, du, vmstat, demidecode, sar, pagesize, etc.

## ****a. Process Management Commands****

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| ps | Shows currently running processes | ps aux |
| pstree | Displays running processes as a tree | pstree |
| nice | Starts a process with a given priority | nice -n 10 ./my\_script.sh |
| renice | Changes the priority of a running process | renice -n 5 -p 1234 |
| kill | Sends a signal to a process (usually to terminate it) | kill -9 1234 |
| pkill | Kills a process by name | pkill firefox |
| killall | Kills all processes with the specified name | killall chrome |
| xkill | Click on a window to forcefully close it (GUI based) | xkill |
| fg | Brings a background process to the foreground | fg |
| bg | Resumes a suspended process in the background | bg |
| pgrep | Finds the PID of a process by name | pgrep ssh |

## ****b. Memory Management Commands****

| **Command** | **Purpose** | **Example** |
| --- | --- | --- |
| free | Shows total, used, and free memory | free -h |
| /proc/meminfo | Displays detailed memory info from the proc filesystem | cat /proc/meminfo |
| top | Interactive real-time process and memory usage | top |
| htop | Enhanced version of top with better UI (may need to install) | htop |
| df | Shows disk space usage | df -h |
| du | Shows space used by files/directories | du -sh \* |
| vmstat | Displays system performance and memory stats | vmstat |
| dmidecode | Shows hardware info including memory details | sudo dmidecode |
| sar | System activity report (needs sysstat package) | sar -r 1 5 |
| pagesize | Displays the size of a memory page | pagesize |

**🔧 STEP 1: Create the Script File**

1. Open a terminal.
2. Create a new .sh file:

bash

CopyEdit

nano simple\_loop.sh

1. Paste this code inside:

bash

CopyEdit

#!/bin/bash

while true

do

echo "Running... Press Ctrl+C to stop."

sleep 5

done

1. Save and exit:

* Press CTRL + O, then Enter to save.
* Press CTRL + X to exit nano.

**✅ STEP 2: Make It Executable**

Run the following command:

bash

CopyEdit

chmod +x simple\_loop.sh

**🚀 STEP 3: Run the Script in Background with Low Priority**

Use the nice command to give it low CPU priority:

bash

CopyEdit

nice -n 10 ./simple\_loop.sh &

* The & puts it in the background.
* You’ll see a process ID (PID) returned.

**🔍 STEP 4: View the Running Process**

Use one of these:

bash

CopyEdit

ps aux | grep simple\_loop.sh

or

bash

CopyEdit

pgrep -fl simple\_loop.sh

or

bash

CopyEdit

top

Look for the script in the list.

**🛑 STEP 5: Kill the Process**

Find its **PID** (Process ID) from the previous step, then run:

bash

CopyEdit

kill <PID>

Or kill it by name:

bash

CopyEdit

pkill -f simple\_loop.sh

You can also stop all background jobs in your current terminal with:

bash

CopyEdit

jobs

kill %1 # If it's job number 1

Let me know if you’re running this on **Ubuntu**, **Fedora**, or **WSL (Windows Subsystem for Linux)** — some steps might vary slightly.

**Experiment 7**

CPU scheduling   
**HOW TO RUN: nano filename.c**

**gcc filename.c -o filename**

**./filename**

Code :

#include <stdio.h>

#include <stdlib.h>

void calc(int bt[], int at[], int ct[], int tat[], int wt[], int n) {

int currentTime = 0;

for (int i = 0; i < n; i++) {

if (at[i] > currentTime) {

currentTime = at[i];

}

ct[i] = currentTime + bt[i];

tat[i] = ct[i] - at[i];

wt[i] = tat[i] - bt[i];

currentTime = ct[i];

}

}

void show(int id[], int bt[], int at[], int ct[], int tat[], int wt[], int n) {

printf("\nProcess\tArrival\tBurst\tCompletion\tTurn-Around\tWaiting\n");

for (int i = 0; i < n; i++) {

printf(" P[%d]\t %d\t %d\t %d\t\t %d\t\t %d\n", id[i], at[i], bt[i], ct[i], tat[i], wt[i]);

}

}

void sortProcesses(int id[], int bt[], int at[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (at[j] > at[j + 1]) {

int temp = at[j];

at[j] = at[j + 1];

at[j + 1] = temp;

temp = bt[j];

bt[j] = bt[j + 1];

bt[j + 1] = temp;

temp = id[j];

id[j] = id[j + 1];

id[j + 1] = temp;

}

}

}

}

int main() {

int n;

printf("Enter the number of processes in your system:\n");

scanf("%d", &n);

if (n <= 0) {

printf("Invalid number of processes. Exiting...\n");

return 1;

}

int id[n], bt[n], at[n], ct[n], tat[n], wt[n];

for (int i = 0; i < n; i++) {

printf("Enter details for process %d:\n", i + 1);

printf("Arrival Time: ");

scanf("%d", &at[i]);

if (at[i] < 0) {

printf("Arrival time cannot be negative. Exiting...\n");

return 1;

}

printf("Burst Time: ");

scanf("%d", &bt[i]);

if (bt[i] <= 0) {

printf("Burst time must be positive. Exiting...\n");

return 1;

}

id[i] = i + 1;

}

sortProcesses(id, bt, at, n);

calc(bt, at, ct, tat, wt, n);

show(id, bt, at, ct, tat, wt, n);

float avg\_tat = 0, avg\_wt = 0;

for (int i = 0; i < n; i++) {

avg\_tat += tat[i];

avg\_wt += wt[i];

}

avg\_tat /= n;

avg\_wt /= n;

printf("\nAverage Turnaround Time: %.2f\n", avg\_tat);

printf("Average Waiting Time: %.2f\n", avg\_wt);

return 0;

}

2. SJF

#include <stdio.h>

#include <stdlib.h>

struct Process {

int id;

int arrival;

int burst;

int completion;

int turnaround;

int waiting;

};

void calculateTimes(struct Process p[], int n) {

int currentTime = 0;

int completed = 0;

int shortest;

while (completed != n) {

shortest = -1;

int minBurst = \_\_INT\_MAX\_\_;

for (int i = 0; i < n; i++) {

if (p[i].arrival <= currentTime && p[i].completion == 0 && p[i].burst < minBurst) {

minBurst = p[i].burst;

shortest = i;

}

}

if (shortest == -1) {

currentTime++;

} else {

p[shortest].completion = currentTime + p[shortest].burst;

p[shortest].turnaround = p[shortest].completion - p[shortest].arrival;

p[shortest].waiting = p[shortest].turnaround - p[shortest].burst;

currentTime = p[shortest].completion;

completed++;

}

}

}

void displayResults(struct Process p[], int n) {

printf("\nProcess\tArrival\tBurst\tCompletion\tTurn-Around\tWaiting\n");

for (int i = 0; i < n; i++) {

printf(" P[%d]\t %d\t %d\t %d\t\t %d\t\t %d\n", p[i].id, p[i].arrival, p[i].burst,

p[i].completion, p[i].turnaround, p[i].waiting);

}

float avg\_tat = 0, avg\_wt = 0;

for (int i = 0; i < n; i++) {

avg\_tat += p[i].turnaround;

avg\_wt += p[i].waiting;

}

avg\_tat /= n;

avg\_wt /= n;

printf("\nAverage Turnaround Time: %.2f\n", avg\_tat);

printf("Average Waiting Time: %.2f\n", avg\_wt);

}

void sortProcesses(struct Process p[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (p[j].arrival > p[j + 1].arrival || (p[j].arrival == p[j + 1].arrival && p[j].burst > p[j +

1].burst)) {

struct Process temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

}

}

int main() {

int n;

printf("Enter the number of processes: ");

scanf("%d", &n);

if (n <= 0) {

printf("Invalid number of processes. Exiting...\n");

return 1;

}

struct Process p[n];

for (int i = 0; i < n; i++) {

printf("Enter details for process %d:\n", i + 1);

printf("Arrival Time: ");

scanf("%d", &p[i].arrival);

if (p[i].arrival < 0) {

printf("Arrival time cannot be negative. Exiting...\n");

return 1;

}

printf("Burst Time: ");

scanf("%d", &p[i].burst);

if (p[i].burst <= 0) {

printf("Burst time must be positive. Exiting...\n");

return 1;

}

p[i].id = i + 1;

p[i].completion = 0; // Initialize completion time to 0

}

sortProcesses(p, n);

calculateTimes(p, n);

displayResults(p, n);

return 0;

}

**Experiment 8**

a. Study of Shell, Types of Shell, Variables and Operators  
b. Execute the following Scripts (at least 6):  
i) Write a shell script to perform arithmetic operations.  
ii) Write a shell script to calculate simple interest.  
iii) Write a shell script to determine the largest among three integer  
numbers.  
iv) Write a shell script to determine if a given year is leap year or not.  
v) Write a shell script to print the multiplication table of given numbers  
using while statement.  
vi) Write a shell script to search whether an element is present in the list  
or not.  
vii) Write a shell script to compare two strings.  
viii) Write a shell script to read and check if the directory / file exists or  
not, if not make the directory / file.  
ix) Write a shell script to implement a menu-driven calculator using case  
statements.  
x) Write a shell script to print following pattern:  
\*  
\*  
\* \*  
\* \* \*  
xi) Write a shell script to perform operations on directory like: display  
name of current directory, display list of directory contents, create  
another directory — write contents on that and copy it to a suitable  
location in your home directory, etc.

to run the script   
1) nano script1.sh

2)paste->ctrl+O->Enter->ctrl+X

3) chmod +x script1.sh

4) to run : sh script1.sh or ./script1.sh

CODE:

**i) Arithmetic operations**

#!/bin/bash

echo "Enter two numbers:"

read a b

echo "Sum: $((a + b))"

echo "Difference: $((a - b))"

echo "Product: $((a \* b))"

echo "Quotient: $((a / b))"

**ii) Simple Interest**

#!/bin/bash

echo "Enter Principal, Rate, and Time:"

read p r t

si=$(( (p \* r \* t) / 100 ))

echo "Simple Interest is: $si"

**iii) Largest among three integers**

#!/bin/bash

echo "Enter three numbers:"

read a b c

if [ $a -ge $b ] && [ $a -ge $c ]; then

echo "$a is largest"

elif [ $b -ge $a ] && [ $b -ge $c ]; then

echo "$b is largest"

else

echo "$c is largest"

fi

**iv) Leap Year Check**

#!/bin/bash

echo "Enter a year:"

read year

if (( (year % 4 == 0 && year % 100 != 0) || year % 400 == 0 )); then

echo "$year is a leap year"

else

echo "$year is not a leap year"

fi

**v) Multiplication Table (while)**

#!/bin/bash

echo "Enter a number:"

read n

i=1

while [ $i -le 10 ]; do

echo "$n \* $i = $((n \* i))"

((i++))

done

**vi) Search element in list**

#!/bin/bash

echo "Enter elements separated by space:"

read -a arr

echo "Enter element to search:"

read key

found=0

for item in "${arr[@]}"; do

if [ "$item" == "$key" ]; then

found=1

break

fi

done

if [ $found -eq 1 ]; then

echo "$key is present in the list"

else

echo "$key is not present in the list"

fi

**vii) Compare two strings**

#!/bin/bash

echo "Enter two strings:"

read str1 str2

if [ "$str1" = "$str2" ]; then

echo "Strings are equal"

else

echo "Strings are not equal"

fi

**viii) Check directory/file and create if not**

#!/bin/bash

echo "Enter file or directory name:"

read name

if [ -e "$name" ]; then

echo "$name exists."

else

echo "$name does not exist. Creating..."

mkdir -p "$name" || touch "$name"

echo "$name created."

fi

**ix) Menu-driven Calculator (case)**

#!/bin/bash

echo "Enter two numbers:"

read a b

echo "Choose operation: 1.Add 2.Subtract 3.Multiply 4.Divide"

read ch

case $ch in

1) echo "Sum = $((a + b))" ;;

2) echo "Difference = $((a - b))" ;;

3) echo "Product = $((a \* b))" ;;

4) echo "Quotient = $((a / b))" ;;

\*) echo "Invalid choice" ;;

esac

**x) Print pattern**

#!/bin/bash

for ((i=1; i<=3; i++)); do

for ((j=1; j<=i; j++)); do

echo -n "\* "

done

echo

done

**xi) Operations on Directory**

#!/bin/bash

echo "Current Directory:"

pwd

echo "Contents:"

ls

echo "Creating new directory 'mydir' and writing a file..."

mkdir -p mydir

echo "This is a sample file" > mydir/sample.txt

echo "Copying to home directory..."

cp -r mydir ~/mydir\_copy

echo "Done."

**Experiment 9**

Execute the following scripts using grep / sed commands:  
i) Write a script using grep command to find the number of words character,  
words and lines in a file.  
ii) Write a script using egrep command to display a list of specific types of  
files in the directory.  
iii) Write a script using sed command to replace all occurrences of a particular  
word in a given file.  
iv) Write a script using sed command to print duplicate lines in input.

How to Run:

**✅ Step 1: Open your terminal**

If you're on **Windows**, do one of the following:

* Use **WSL (Ubuntu)** if installed.
* Or install something like **Git Bash**, or use a virtual Linux environment.

**✅ Step 2: Create the script files**

Example for the first script:

bash

CopyEdit

nano count\_file\_stats.sh

Paste the script content in the editor, then press:

* Ctrl + O to save
* Enter to confirm filename
* Ctrl + X to exit

Repeat this for each script, or create them all in a single session.

**✅ Step 3: Make the scripts executable**

bash

CopyEdit

chmod +x count\_file\_stats.sh

chmod +x list\_file\_types.sh

chmod +x replace\_word.sh

chmod +x print\_duplicates.sh

Or do it all at once:

bash

CopyEdit

chmod +x \*.sh

**✅ Step 4: Run the scripts**

Make sure you're in the directory where the script exists.

**Examples:**

bash

CopyEdit

./count\_file\_stats.sh sample.txt

./list\_file\_types.sh

./replace\_word.sh oldword newword sample.txt

./print\_duplicates.sh sample.txt

Make sure to have the **sample file (like sample.txt)** present in the same directory or provide the correct path.

**💡 Pro Tip (Run All at Once)**

To run all scripts in one go:

bash

CopyEdit

./count\_file\_stats.sh sample.txt

./list\_file\_types.sh

./replace\_word.sh oldword newword sample.txt

./print\_duplicates.sh sample.txt

Or combine them into a single run\_all.sh script:

bash

CopyEdit

#!/bin/bash

./count\_file\_stats.sh sample.txt

./list\_file\_types.sh

./replace\_word.sh oldword newword sample.txt

./print\_duplicates.sh sample.txt

Then run:

bash

CopyEdit

chmod +x run\_all.sh

./run\_all.sh

**CODE:**

Here are **4 shell scripts** that use grep, egrep, and sed as requested:

### ✅ i) Script using grep to count ****characters, words, and lines**** in a file

**Filename:** count\_file\_stats.sh

#!/bin/bash

# Usage: ./count\_file\_stats.sh filename.txt

if [ $# -ne 1 ]; then

echo "Usage: $0 filename"

exit 1

fi

echo "Characters: $(grep -o . "$1" | wc -l)"

echo "Words: $(grep -oE '\w+' "$1" | wc -l)"

echo "Lines: $(grep -c '' "$1")"

### ✅ ii) Script using egrep to list ****specific types of files**** (e.g., .txt, .sh)

**Filename:** list\_file\_types.sh

#!/bin/bash

# Usage: ./list\_file\_types.sh

echo "Listing .txt and .sh files:"

ls | egrep "\.txt$|\.sh$"

### ✅ iii) Script using sed to ****replace all occurrences**** of a word

**Filename:** replace\_word.sh

#!/bin/bash

# Usage: ./replace\_word.sh oldword newword filename.txt

if [ $# -ne 3 ]; then

echo "Usage: $0 oldword newword filename"

exit 1

fi

sed "s/$1/$2/g" "$3"

💡 Use -i if you want to replace in the file: sed -i "s/$1/$2/g" "$3"

### ✅ iv) Script using sed to ****print duplicate lines**** from input

**Filename:** print\_duplicates.sh

#!/bin/bash

# Usage: ./print\_duplicates.sh filename.txt

if [ $# -ne 1 ]; then

echo "Usage: $0 filename"

exit 1

fi

sed -n 'N;/^\(.\*\)\n\1$/p' "$1"

🔁 This prints **consecutive duplicate lines**. If you want **all duplicate lines regardless of position**, better to use:

sort "$1" | uniq -d

Would you like me to zip these scripts and send them as a downloadable file?