
1.to find .txt and .log files in directory

```
logs_cnt=$(find . -maxdepth 1 -type f -name '*.logs' | wc -l)
txt_cnt=$(find . -maxdepth 1 -type f -name '*.txt'  | wc -l)

echo ".logs files : $logs_cnt"
echo ".txt  files : $txt_cnt”
```

1.b. Recursively

```
log_count=$(find . -type f -name "*.log" | wc -l)
echo "logfiles:$log_count"

txt_count=$(find . -type f -name "*.txt" | wc -l)
echo "txtfiles:$txt_count”
```

---

2 to find the . Files in $DIR1 but NOT in $DIR2:”

```
#!/bin/bash

DIR1="dir1"
DIR2="dir2"

echo "Files in $DIR1 but NOT in $DIR2:"

for file in "$DIR1"/*; do
  filename=$(basename "$file")
  if [[ ! -e "$DIR2/$filename" ]]; then
    echo "$filename"
  fi
```

- done

---

3. From a file.txt find comman repeated words

File-Wide Duplicate Word Counter

```
#!/bin/bash
file="file.txt"

# Check if the file exists
if [[ ! -f "$file" ]]; then
  echo "Error: $file not found!"
```

```
  exit 1
fi

# Extract repeated words and store in a variable
common_words=$(grep -oE '\w+' "$file" | sort | uniq -c | awk '$1 > 1 {print $2}')

# Print the result
echo "Repeated words:"
echo "$common_words”
```

Line-by-Line Duplicate Word Detector

```
#!/bin/bash
while read -r line; do
  words=($line)
  dup=$(printf "%s\n" "${words[@]}" | sort | uniq -d)
  if [[ -n $dup ]]; then
    echo "$line"
  fi
done < input.txt
```

4.Check if file size of file exceeds then the given thresholdwith output dhould be file timestamp threshold

```
#!/bin/bash

# Input arguments
logfile="$1"        # Path to the log file
threshold_mb="$2"   # Threshold in MB

# Get file size in MB (rounded)
filesize_mb=$(du -m "$logfile" | cut -f1)

# Get current timestamp
timestamp=$(date +%Y%m%d_%H%M%S)

# Check if file size exceeds the threshold
if [ "$filesize_mb" -gt "$threshold_mb" ]; then
    echo "${logfile##*/}_${filesize_mb}MB_${timestamp}"
fi

Emer can also use
du -m "$logfile" | awk '{print $1}'
```

```
chmod +x check_log_size.sh
./check_log_size.sh /var/log/syslog 10
```

5.download the 100 repository from GitHub output should be github_backup_2025-06-18

```
#!/bin/bash

# === Configuration ===
GITHUB_USER="your-github-username-or-org"   # ← Replace this
BACKUP_DIR="github_backup_$(date +%F)"      # Example: github_backup_2025-06-18

# === Create backup directory ===
mkdir -p "$BACKUP_DIR"
cd "$BACKUP_DIR" || exit 1

# === Fetch repo clone URLs using GitHub API ===
echo "[*] Fetching repositories for $GITHUB_USER..."

REPO_URLS=$(curl -s
"https://api.github.com/users/$GITHUB_USER/repos?per_page=100" | \
  grep -oP '"clone_url": "\K(.*?)(?=")')

# === Clone each repo (full clone, not mirror) ===
for repo in $REPO_URLS; do
    echo "[*] Cloning: $repo"
    git clone "$repo"
done

# === Go up and compress the backup ===
cd ..
tar -czf "${BACKUP_DIR}.tar.gz" "$BACKUP_DIR"
echo "[✔] Backup complete: ${BACKUP_DIR}.tar.gz"

# === Optional: Remove uncompressed folder ===
# rm -rf "$BACKUP_DIR"
```

```
#!/bin/bash
# Script Name: backup_tar.sh
# Description: This script creates a compressed .tar.gz archive of a directory
#              named 'backup'. It includes a timestamp in the filename.
#              If the directory doesn't exist, it will be created.
# ----------------------------------------------------------------------------------------------------------------
```

6.Backup archive created: $TAR_NAME”

```
# Set the name of the directory to back up
DIR_NAME="backup"

# Check if the directory exists
if [ ! -d "$DIR_NAME" ]; then
    echo "📁 Directory '$DIR_NAME' does not exist. Creating it..."
    mkdir "$DIR_NAME"
    echo "✅ Directory '$DIR_NAME' created."
else
    echo "📁 Directory '$DIR_NAME' already exists. Proceeding to compress."
fi

# Get the current date and time in the format YYYY-MM-DD_HH-MM-SS
TIMESTAMP=$(date +%F_%H-%M-%S)

# Construct the name of the tar.gz file using the directory name and timestamp
TAR_NAME="${DIR_NAME}_${TIMESTAMP}.tar.gz"

# Create the tar.gz arch5ive using the tar command
# -c : create new archive
# -z : filter the archive through gzip for compression
# -v : verbosely list files being processed
# -f : use archive file name
tar -czvf "$TAR_NAME" "$DIR_NAME"

# Confirm the archive was created
echo "✅ Backup archive created: $TAR_NAME”
.

.#####################################
```

7.kill high usage proceess
Checks every 10 minutes
Kills any process using >80% CPU
Logs the action with timestamp

```
#!/bin/bash

log="/home/youruser/kill_cpu.log"

while true; do
  ps -eo pid,%cpu,comm --no-headers | awk '$2 > 80' | while read pid cpu cmd; do
    echo "$(date '+%F %T') - Killed $pid ($cmd) using $cpu% CPU" >> "$log"
```

```
    kill -9 "$pid"
  done
  sleep 600  # wait 10 minutes
done
```

.###################################

8. Keep only laSt 3 backup

```
#!/bin/bash

# === CONFIGURATION ===
SRC="/home/ubuntu/myapp"          # Source directory to back up
DEST="/home/ubuntu/backups"       # Destination for backup files
TS=$(date +%F_%H-%M-%S)           # Timestamp format: YYYY-MM-DD_HH-MM-SS
FILE="backup_$TS.tar.gz"

# === CREATE BACKUP ===
mkdir -p "$DEST"
tar -czf "$DEST/$FILE" "$SRC"

# === KEEP ONLY LAST 3 BACKUPS ===
cd "$DEST" || exit 1
ls -tp backup_*.tar.gz | grep -v '/$' | tail -n +4 | xargs -r rm --

echo "Backup complete: $FILE”
```

-

```
#!/bin/bash

# Create archive with time-based name
ARCHIVE_NAME="archive_$(date +%H%M%S).zip"

# Find files older than 3 minutes and archive them
find . -type f -mmin +3 | tee /tmp/file_list.txt | xargs zip "$ARCHIVE_NAME"

# If zip successful, delete original files
if [ $? -eq 0 ]; then
    cat /tmp/file_list.txt | xargs rm -f
    echo "Archived and deleted old files."
else
    echo "Failed to create archive."
fi
```

```
# Clean up
rm -f /tmp/file_list.txt
```

```
#!/bin/bash

process_name="nginx"
start_process="sudo systemctl start nginx"

if ps aux | grep -v grep | grep -w "$process_name" > /dev/null
then
  echo "Process $process_name is running..."
else
  echo "Process $process_name is not running... so starting $process_name"
  $start_process
fi


dir="dir1"
pattern="error"

result=$(grep -ri "$pattern" "$dir")
echo "$result"

dir1/
├── file1.txt  (contains: "Some ERROR happened")
├── file2.txt  (contains: "All good")
```

Script: Check if Website Returns HTTP 200

```
#!/bin/bash

site="https://google.com"
error_log="error.log"

# Clear previous error log
> "$error_log"

# Get the HTTP status code
status_code=$(curl -s -o /dev/null -w "%{http_code}" "$site")
```

```
# Check for HTTP 200 OK
if [[ "$status_code" -eq 200 ]]; then
   echo "$site is UP (HTTP $status_code)"
else
   echo "$site is DOWN or returned HTTP $status_code" | tee -a "$error_log"
fi
```

Bash Script Using ping (Single Website)

```
#!/bin/bash

site="google.com"
error_log="error.log"

# Clear previous error log
> "$error_log"

# Ping once (-c 1), suppress output (&> /dev/null)
if ping -c 1 "$site" &> /dev/null; then
   echo "$site is reachable"
else
   echo "$site is unreachable" | tee -a "$error_log"
fi
```

ping -c 1: Sends one ping packet to the domain.

&> /dev/null: Silences all output (standard and error).

if ...; then: Checks if ping command returns success (exit code 0).

tee -a error.log: Prints to terminal and appends to error.log.