

## ■ Linux Administrator

---

Every Module · All Scripts · Full Exercises · Expected Outputs

7 Modules • 5 Real-World Projects • Production Patterns

Designed for Linux Administrators

## Module 1 Script Foundations

### 1. The Shebang Line

The very first line of every bash script must be:

```
#!/bin/bash
```

This tells the system which interpreter to use to run the script. Without it, the script may behave unpredictably depending on the shell the user is running.

### 2. Basic Script Structure

```
#!/bin/bash

# =====

# Script Name: system_info.sh

# Description: Displays basic system information

# Author: Your Name

# Date: 2024-01-01

# Version: 1.0

# =====

# --- Variables ---

HOSTNAME=$(hostname)

OS=$(cat /etc/os-release | grep PRETTY_NAME | cut -d= -f2)

UPTIME=$(uptime -p)

DISK=$(df -h / | awk 'NR==2 {print $5}')

MEMORY=$(free -h | awk '/Mem:/ {print $3 "/" $2}')

# --- Main Logic ---

echo "==== System Information ====="

echo "Hostname   : $HOSTNAME"

echo "OS          : $OS"

echo "Uptime      : $UPTIME"

echo "Disk Used   : $DISK"

echo "Memory      : $MEMORY"
```

```
echo "====="
```

### 3. Variables — The Rules

```
# Defining variables — NO spaces around =

NAME="LinuxAdmin"

AGE=5

# Accessing variables — always use $

echo $NAME

echo "I have $AGE years of experience"

# Command output stored in a variable

CURRENT_USER=$(whoami)

echo "Running as: $CURRENT_USER"

# Read-only variable (like a constant)

readonly MAX_RETRIES=3

# Best practice — use UPPERCASE for variable names
```

■■ *Never put spaces around = when assigning variables. NAME = 'test' will fail.*

### 4. Taking User Input

```
#!/bin/bash

# Basic input

read -p "Enter username to create: " USERNAME

echo "You entered: $USERNAME"

# Silent input (for passwords)

read -sp "Enter password: " PASSWORD

echo "" # move to new line after silent input

# Input with a timeout (useful in automated scripts)

read -t 10 -p "Enter your choice (10 sec timeout): " CHOICE
```

### 5. Making a Script Executable & Running It

```
# Give execute permission
```

```
chmod +x system_info.sh

# Run it

./system_info.sh

# Or run with bash explicitly

bash system_info.sh

# Run as root if needed

sudo ./system_info.sh
```

## 6. Special Built-in Variables

```
$0      # Name of the script itself

$1      # First argument passed to the script

$2      # Second argument

$#      # Total number of arguments passed

$@      # All arguments

$?      # Exit code of the last command (0 = success)

$$      # PID of the current script

$USER   # Current user running the script
```

Example using arguments:

```
#!/bin/bash

# Usage: ./create_user.sh john

USERNAME=$1

if [ -z "$USERNAME" ]; then

    echo "Usage: $0 <username>"

    exit 1

fi

echo "Creating user: $USERNAME"

useradd $USERNAME

echo "Done!"
```

### ■ ■ Exercise

1. Write a script called `server_info.sh` that:
2. Accepts a name as an argument (e.g. `./server_info.sh John`)
3. Greets the user by name
4. Displays: hostname, current logged-in user, current date & time, and how much disk space is used on `/`
5. If no name is passed, it should print a usage message and exit

### ■ Solution — `server_info.sh`

```
#!/bin/bash

# =====

# Script Name: server_info.sh

# Description: Displays server information

# Usage      : ./server_info.sh <your_name>

# =====

# --- Check if an argument was passed ---

if [ -z "$1" ]; then

    echo "Usage: $0 <your_name>"

    exit 1

fi

# --- Variables ---

NAME=$1

HOSTNAME=$(hostname)

CURRENT_USER=$(whoami)

DATE_TIME=$(date "+%Y-%m-%d %H:%M:%S")

DISK_USED=$(df -h / | awk 'NR==2 {print $3 " used out of " $2 " ( " $5 " )"}')

# --- Main Output ---

echo "===== "

echo "  Welcome, $NAME!"

echo "===== "

echo "Hostname      : $HOSTNAME"
```

```
echo "Logged in as : $CURRENT_USER"

echo "Date & Time   : $DATE_TIME"

echo "Disk Usage    : $DISK_USED"

echo "====="
```

How to run:

```
chmod +x server_info.sh

./server_info.sh John

./server_info.sh  # Test usage message
```

Expected Output:

```
=====

Welcome, John!

=====

Hostname      : prod-server-01
Logged in as  : john
Date & Time   : 2024-01-15 10:35:22
Disk Usage    : 12G used out of 50G ( 24% )

=====
```

■ *Key Takeaways: Always start with #!/bin/bash, validate arguments with -z, use \$() to capture command output, use UPPERCASE variable names, exit 1 on errors.*

## Module 2      Conditionals & Logic

### 1. The if/elif/else Structure

```
# Basic syntax

if [ condition ]; then

    # do something

elif [ another_condition ]; then

    # do something else

else

    # fallback

fi
```

■■ Always have spaces inside the brackets [ condition ]. Missing spaces will cause errors.

### 2. File & Directory Test Operators

```
[ -f /etc/passwd ]      # True if file EXISTS

[ -d /var/log ]         # True if DIRECTORY exists

[ -r /etc/shadow ]      # True if file is READABLE

[ -w /tmp/test ]        # True if file is WRITABLE

[ -x /usr/bin/python ]  # True if file is EXECUTABLE

[ -s /var/log/syslog ]  # True if file is NOT EMPTY

[ -e /etc/hosts ]       # True if path EXISTS (file or dir)
```

Real example:

```
#!/bin/bash

LOGFILE="/var/log/app.log"

if [ -f "$LOGFILE" ]; then

    echo "Log file found. Size: $(du -sh $LOGFILE | cut -f1)"

else

    echo "Log file not found. Creating it..."

    touch "$LOGFILE"

    echo "Log file created successfully."
```

```
fi
```

### 3. String Test Operators

```
[ -z "$VAR" ]      # True if string is EMPTY
[ -n "$VAR" ]      # True if string is NOT empty
[ "$A" = "$B" ]    # True if strings are EQUAL
[ "$A" != "$B" ]   # True if strings are NOT equal
```

Real example — Service status check:

```
#!/bin/bash

SERVICE=$1

if [ -z "$SERVICE" ]; then
    echo "Usage: $0 <service_name>"
    exit 1
fi

STATUS=$(systemctl is-active $SERVICE)

if [ "$STATUS" = "active" ]; then
    echo "$SERVICE is running"
else
    echo "$SERVICE is NOT running. Attempting restart..."
    systemctl restart $SERVICE
    echo "Restart triggered for $SERVICE"
fi
```

### 4. Numeric Test Operators

```
[ $A -eq $B ]      # Equal
[ $A -ne $B ]      # Not equal
[ $A -gt $B ]      # Greater than
[ $A -lt $B ]      # Less than
[ $A -ge $B ]      # Greater than or equal
[ $A -le $B ]      # Less than or equal
```

Real example — Disk usage alert:

```
#!/bin/bash

THRESHOLD=80

DISK_USAGE=$(df / | awk 'NR==2 {print $5}' | tr -d '%')

if [ $DISK_USAGE -ge $THRESHOLD ]; then

    echo "WARNING: Disk usage is at ${DISK_USAGE}% - above threshold of ${THRESHOLD}%"

else

    echo "Disk usage is at ${DISK_USAGE}% - within safe limits"

fi
```

## 5. Combining Conditions

```
# AND - both conditions must be true

if [ $DISK -gt 80 ] && [ -f "/etc/alert.conf" ]; then

    echo "Alert config exists and disk is high"

fi

# OR - at least one condition must be true

if [ "$USER" = "root" ] || [ "$USER" = "admin" ]; then

    echo "Privileged user detected"

fi
```

## 6. The case Statement

```
#!/bin/bash

ACTION=$1

case "$ACTION" in

    start)

        echo "Starting the service..."

        systemctl start nginx

        ;;

    stop)

        echo "Stopping the service..."

        systemctl stop nginx
```

```
;;

restart)

    echo "Restarting the service..."

    systemctl restart nginx

;;

status)

    systemctl status nginx

;;

*)

    echo "Usage: $0 {start|stop|restart|status}"

    exit 1

;;

esac
```

■■ The \*) at the end is the default case — like else. Always include it.

## 7. Checking if Running as Root

```
#!/bin/bash

if [ "$EUID" -ne 0 ]; then

    echo "This script must be run as root. Use sudo."

    exit 1

fi

echo "Running as root. Proceeding..."
```

### ■■ Exercise

1. Write a script called `service_check.sh` that:
2. Checks if it is being run as root — if not, exit with a message
3. Accepts a service name as an argument — if none given, show usage and exit
4. Checks if the service is active — if yes, print a success message
5. If the service is not active, attempt to restart it
6. After restarting, check again — if now active print success, if still failing print a critical alert
7. Bonus — check if disk usage on `/` is above 85% and warn the user at the end

## ■ Solution — service\_check.sh

```
#!/bin/bash

# =====

# Script Name: service_check.sh

# Description: Checks and recovers a service,

#              and monitors disk usage

# Usage       : sudo ./service_check.sh <service_name>

# =====

# --- Check if running as root ---

if [ "$EUID" -ne 0 ]; then

    echo "This script must be run as root. Use sudo."

    exit 1

fi

# --- Check if argument was passed ---

if [ -z "$1" ]; then

    echo "Usage: $0 <service_name>"

    exit 1

fi

# --- Variables ---

SERVICE=$1

DISK_THRESHOLD=85

DISK_USAGE=$(df / | awk 'NR==2 {print $5}' | tr -d '%')

# --- Check service status ---

STATUS=$(systemctl is-active $SERVICE)

if [ "$STATUS" = "active" ]; then

    echo "$SERVICE is running normally."

else

    echo "$SERVICE is NOT running. Attempting restart..."

    systemctl restart $SERVICE
```

```
# --- Re-check after restart ---

STATUS=$(systemctl is-active $SERVICE)

if [ "$STATUS" = "active" ]; then

    echo "$SERVICE successfully restarted."

else

    echo "CRITICAL: $SERVICE failed to restart. Manual intervention required!"

    exit 1

fi

fi

# --- Bonus: Disk usage check ---

if [ $DISK_USAGE -ge $DISK_THRESHOLD ]; then

    echo "WARNING: Disk usage is at ${DISK_USAGE}% - above ${DISK_THRESHOLD}% threshold!"

else

    echo "Disk usage is at ${DISK_USAGE}% - within safe limits."

fi
```

How to run:

```
chmod +x service_check.sh

sudo ./service_check.sh nginx

sudo ./service_check.sh sshd

sudo ./service_check.sh    # Test usage message
```

Expected Output:

```
sshd is running normally.

WARNING: Disk usage is at 87% - above 85% threshold!
```

## Module 3    Loops

Loops let you automate repetitive tasks across multiple files, users, servers, or log entries. This is where bash scripting becomes truly powerful for sysadmins.

### 1. The for Loop

```
# Basic syntax

for VARIABLE in list; do

    # do something

done

# Loop over a simple list

for SERVICE in nginx sshd cron firewallld; do

    STATUS=$(systemctl is-active $SERVICE)

    echo "$SERVICE : $STATUS"

done

# Loop over files

for LOGFILE in /var/log/*.log; do

    SIZE=$(du -sh $LOGFILE | cut -f1)

    echo "$LOGFILE - Size: $SIZE"

done

# Loop over command output

for MOUNT in $(df -h | awk 'NR>1 {print $6}'); do

    echo "Checking: $MOUNT"

done
```

### 2. The while Loop — Retry Logic

```
#!/bin/bash

SERVICE="nginx"

MAX_RETRIES=3

COUNT=0
```

```
while [ $COUNT -lt $MAX_RETRIES ]; do

    STATUS=$(systemctl is-active $SERVICE)

    if [ "$STATUS" = "active" ]; then

        echo "$SERVICE is running."

        break    # exit the loop immediately

    else

        COUNT=$((COUNT + 1))

        echo "Attempt $COUNT: $SERVICE not running. Retrying in 5 seconds..."

        sleep 5

        systemctl restart $SERVICE

    fi

done

if [ $COUNT -eq $MAX_RETRIES ]; then

    echo "CRITICAL: $SERVICE failed after $MAX_RETRIES attempts!"

fi
```

Reading a file line by line:

```
#!/bin/bash

# Process a list of servers from a file

while IFS= read -r SERVER; do

    echo "Pinging $SERVER..."

    ping -c 1 $SERVER &>/dev/null && echo "$SERVER is UP" || echo "$SERVER is DOWN"

done < /etc/serverlist.txt
```

■ *Reading files line-by-line with 'while read' is one of the most used patterns in real sysadmin scripting.*

### 3. The until Loop

```
#!/bin/bash

# Wait until a service comes up

SERVICE="nginx"

until [ "$(systemctl is-active $SERVICE)" = "active" ]; do

    echo "Waiting for $SERVICE to start..."
```

```
    sleep 3

done

echo "$SERVICE is now active!"
```

## 4. Loop Control — break and continue

```
#!/bin/bash

for USER in john mary root guest admin; do

    # Skip root account

    if [ "$USER" = "root" ]; then

        echo "Skipping root..."

        continue

    fi

    # Stop if we hit guest

    if [ "$USER" = "guest" ]; then

        echo "Guest found. Stopping."

        break

    fi

    echo "Processing user: $USER"

done
```

## 5. Arithmetic in Loops

```
# C-style for loop

for (( i=1; i<=5; i++ )); do

    echo "Iteration: $i"

done

# Countdown

COUNT=10

while [ $COUNT -gt 0 ]; do

    echo "Countdown: $COUNT"

    COUNT=$((COUNT - 1))

done
```

```
    sleep 1

done

echo "Launch!"
```

## 6. Real-World Example — Multi-Server Health Check

```
#!/bin/bash

# =====

# Script Name: health_check.sh

# Description: Checks health of multiple servers

# =====

SERVERS=("web01" "web02" "db01" "db02" "cache01")

SERVICES=("nginx" "sshd" "cron")

REPORT="/tmp/health_report_$(date +%Y%m%d).txt"

echo "==== Server Health Report - $(date) ===== | tee $REPORT"

for SERVER in "${SERVERS[@]}; do

    echo "" | tee -a $REPORT

    echo "--- Checking: $SERVER ---" | tee -a $REPORT

    # Ping check

    ping -c 1 $SERVER &>/dev/null

    if [ $? -eq 0 ]; then

        echo "$SERVER is reachable" | tee -a $REPORT

    else

        echo "$SERVER is UNREACHABLE" | tee -a $REPORT

        continue # skip service check if server is down

    fi

    # Service check via SSH

    for SERVICE in "${SERVICES[@]}; do

        STATUS=$(ssh -o ConnectTimeout=5 $SERVER "systemctl is-active $SERVICE" 2>/dev/null)

        if [ "$STATUS" = "active" ]; then
```

```

        echo "  $SERVICE is running" | tee -a $REPORT

    else

        echo "  $SERVICE is NOT running" | tee -a $REPORT

    fi

done

done

echo "===== Report saved to $REPORT ====="
```

## ■ Exercise

1. Write a script called `user_audit.sh` that:
2. Loops through all users in `/etc/passwd` with a shell of `/bin/bash` (real users)
3. For each user, checks if their home directory exists
4. Checks if the home directory is empty or has files in it
5. Prints a clean report showing each user, home directory status, and file count
6. At the end prints a summary of total users checked

## ■ Solution — `user_audit.sh`

```

#!/bin/bash

# =====

# Script Name: user_audit.sh

# Description: Audits real users and their
#              home directory status

# Usage      : ./user_audit.sh

# =====

REPORT="/tmp/user_audit_$(date +%Y%m%d).txt"

TOTAL=0; MISSING=0; EMPTY=0; POPULATED=0

echo "===== User Audit Report - $(date) =====" | tee $REPORT

echo "" | tee -a $REPORT

# Loop through users with /bin/bash shell

while IFS=: read -r USERNAME PASSWORD UID GID COMMENT HOME SHELL; do
```

```
if [ "$SHELL" = "/bin/bash" ]; then

    TOTAL=$((TOTAL + 1))

    echo "User: $USERNAME" | tee -a $REPORT

    echo "  Home: $HOME" | tee -a $REPORT

    if [ ! -d "$HOME" ]; then

        echo "  Status: Home directory MISSING" | tee -a $REPORT

        MISSING=$((MISSING + 1))

    else

        FILE_COUNT=$(ls -A "$HOME" | wc -l)

        if [ "$FILE_COUNT" -eq 0 ]; then

            echo "  Status: Home directory is EMPTY" | tee -a $REPORT

            EMPTY=$((EMPTY + 1))

        else

            echo "  Status: Home has $FILE_COUNT file(s)" | tee -a $REPORT

            POPULATED=$((POPULATED + 1))

        fi

    fi

    echo "" | tee -a $REPORT

fi

done < /etc/passwd

echo "==== Summary =====" | tee -a $REPORT

echo "Total Users Checked : $TOTAL" | tee -a $REPORT

echo "Populated Homes      : $POPULATED" | tee -a $REPORT

echo "Empty Homes          : $EMPTY" | tee -a $REPORT

echo "Missing Homes        : $MISSING" | tee -a $REPORT

echo "Report saved to       : $REPORT" | tee -a $REPORT
```

Expected Output:

```
===== User Audit Report - 2024-01-15 =====

User: root
```

```
Home: /root

Status: Home has 12 file(s)

User: john

Home: /home/john

Status: Home has 45 file(s)

User: mary

Home: /home/mary

Status: Home directory is EMPTY

===== Summary =====

Total Users Checked : 3

Populated Homes      : 2

Empty Homes          : 1

Missing Homes        : 0
```

## Module 4      Functions

Functions are what transform a basic script into clean, maintainable, production-grade code. They allow you to write logic once and reuse it anywhere in your script.

### 1. Basic Function Syntax

```
# Define a function

function_name() {

    # code here

}

# Call a function

function_name

# Simple example

greet_user() {

    echo "Hello, $1! Welcome to the server."

}

greet_user "John"

greet_user "Mary"
```

■■ Always define functions *BEFORE* you call them. Bash reads top to bottom.

### 2. Functions with Arguments

```
#!/bin/bash

check_service() {

    SERVICE=$1

    STATUS=$(systemctl is-active $SERVICE)

    if [ "$STATUS" = "active" ]; then

        echo "$SERVICE is running"

    else

        echo "$SERVICE is NOT running"

    fi

}
```

```
}

# Call the function for multiple services

check_service nginx

check_service sshd

check_service cron
```

### 3. Return Values & Exit Codes

```
#!/bin/bash

is_service_running() {

    SERVICE=$1

    STATUS=$(systemctl is-active $SERVICE 2>/dev/null)

    if [ "$STATUS" = "active" ]; then

        return 0    # success

    else

        return 1    # failure

    fi

}

# Use the return value

if is_service_running nginx; then

    echo "nginx is UP"

else

    echo "nginx is DOWN"

fi

# Returning text from a function – use echo + $()

get_disk_usage() {

    df / | awk 'NR==2 {print $5}' | tr -d '%'

}

USAGE=$(get_disk_usage)

echo "Current disk usage: ${USAGE}%"
```

## 4. Local Variables — Always Use Them

```
#!/bin/bash

get_user_count() {

    local USER_COUNT=$(grep "/bin/bash" /etc/passwd | wc -l)

    echo $USER_COUNT

}

check_disk() {

    local THRESHOLD=$1

    local USAGE=$(df / | awk 'NR==2 {print $5}' | tr -d '%')

    if [ $USAGE -ge $THRESHOLD ]; then

        echo "Disk at ${USAGE}% - above ${THRESHOLD}% threshold"

        return 1

    else

        echo "Disk at ${USAGE}% - within safe limits"

        return 0

    fi

}

USERS=$(get_user_count)

echo "Total bash users: $USERS"

check_disk 80
```

## 5. The Logger Function — Production Essential

```
#!/bin/bash

LOGFILE="/var/log/myscript.log"

log() {

    local LEVEL=$1

    local MESSAGE=$2

    local TIMESTAMP=$(date "+%Y-%m-%d %H:%M:%S")

    echo "[$TIMESTAMP] [$LEVEL] $MESSAGE" | tee -a $LOGFILE
```

```
}

log "INFO"      "Script started"

log "INFO"      "Checking services..."

log "WARNING"   "Disk usage is high"

log "ERROR"     "Service failed to start"

log "INFO"      "Script completed"
```

Output:

```
[2024-01-15 10:30:01] [INFO] Script started

[2024-01-15 10:30:01] [INFO] Checking services...

[2024-01-15 10:30:02] [WARNING] Disk usage is high

[2024-01-15 10:30:03] [ERROR] Service failed to start

[2024-01-15 10:30:03] [INFO] Script completed
```

## 6. Real-World Example — Full Structured Script with Functions

```
#!/bin/bash

# =====

# Script Name: system_health.sh

# Description: Full system health check using

#               modular functions

# Usage       : sudo ./system_health.sh

# =====

LOGFILE="/var/log/system_health.log"

DISK_THRESHOLD=85

SERVICES=("nginx" "sshd" "cron")

# --- Functions ---

log() {

    local LEVEL=$1; local MESSAGE=$2

    local TIMESTAMP=$(date "+%Y-%m-%d %H:%M:%S")

    echo "[$TIMESTAMP] [$LEVEL] $MESSAGE" | tee -a $LOGFILE

}
```

```
check_root() {

    if [ "$EUID" -ne 0 ]; then

        echo "Run as root. Use sudo."; exit 1

    fi

}

check_disk() {

    local USAGE=$(df / | awk 'NR==2 {print $5}' | tr -d '%')

    if [ $USAGE -ge $DISK_THRESHOLD ]; then

        log "WARNING" "Disk usage is at ${USAGE}%"

    else

        log "INFO" "Disk usage is at ${USAGE}% - OK"

    fi

}

check_memory() {

    local TOTAL=$(free -m | awk '/Mem:/ {print $2}')

    local USED=$(free -m | awk '/Mem:/ {print $3}')

    local PERCENT=$(( USED * 100 / TOTAL ))

    if [ $PERCENT -ge 90 ]; then

        log "WARNING" "Memory at ${PERCENT}% - critical!"

    else

        log "INFO" "Memory at ${PERCENT}% - OK"

    fi

}

is_running() {

    [ "$(systemctl is-active $1)" = "active" ]

}

check_services() {

    for SERVICE in "${SERVICES[@]}; do

        if is_running $SERVICE; then
```

```
        log "INFO" "$SERVICE is running"

    else

        log "ERROR" "$SERVICE is NOT running -- restarting"

        systemctl restart $SERVICE

        if is_running $SERVICE; then

            log "INFO" "$SERVICE restarted successfully"

        else

            log "ERROR" "$SERVICE failed to restart!"

        fi

    fi

done

}

# --- Main ---

check_root

log "INFO" "==== Health Check Started ====="

check_disk

check_memory

check_services

log "INFO" "==== Health Check Completed ====="
```

## ■ ■ Exercise

1. Write a script called `system_health.sh` using functions that:
2. Has a `log()` function with INFO/WARN/ERROR levels
3. Has a `check_root()` function
4. Has a `check_disk()` function with configurable threshold
5. Has a `check_memory()` function
6. Has an `is_running()` and `check_services()` function for a list of services
7. Has a clean Main section that calls all functions in order

■ **Key Takeaways:** Always use 'local' for variables inside functions. Use 'return 0/1' for success/failure. Use `echo + $()` to return text. Put log functions in every production script.

## Module 5 Working with Files & Text

As a sysadmin, most of your automation will involve parsing logs, config files, and command outputs. Mastering these tools makes you extremely powerful.

### 1. grep — Searching Text

```
grep -i 'error' /var/log/syslog      # Case insensitive
grep -r 'failed' /var/log/          # Recursive search in directory
grep -v 'INFO' /var/log/app.log     # Invert — show lines NOT matching
grep -n 'error' /var/log/app.log    # Show line numbers
grep -c 'error' /var/log/app.log    # Count matching lines
grep -l 'error' /var/log/*.log      # Show only filenames that match
grep -A3 'error' /var/log/app.log   # Show 3 lines AFTER match
grep -B3 'error' /var/log/app.log   # Show 3 lines BEFORE match
grep -E 'error|failed|critical' file # Extended regex — multiple patterns
```

Real sysadmin examples:

```
#!/bin/bash

LOGFILE="/var/log/auth.log"

# Find all failed SSH login attempts

echo "=== Failed SSH Logins ==="

grep "Failed password" $LOGFILE | awk '{print $11}' | sort | uniq -c | sort -rn

# Find which users had successful logins

echo "=== Successful Logins ==="

grep "Accepted password" $LOGFILE | awk '{print $9}' | sort | uniq -c

# Count today's errors

TODAY=$(date +"%b %d")

ERROR_COUNT=$(grep "$TODAY" $LOGFILE | grep -c 'error')

echo "Errors today: $ERROR_COUNT"
```

### 2. awk — Column & Field Processing

```
# Basic syntax — process field number N
```

```

awk '{print $N}' filename

# Key built-in variables

# $0=entire line  $1=first field  $NF=last field

# NR=line number  NF=number of fields  FS=field separator

# Print specific columns from df output

df -h | awk '{print $1, $5}'

# Print lines where field 3 is greater than 80

df / | awk 'NR>1 && $5+0 > 80 {print $1, "is above 80%:", $5}'

# Custom field separator – parse /etc/passwd

awk -F: '{print $1, $6}' /etc/passwd

# Print lines matching a pattern

awk '/error/ {print NR: "$0}' /var/log/app.log

# Sum a column – total memory of all processes

ps aux | awk '{sum += $6} END {print "Total Memory Used:", sum/1024, "MB"}'
```

Real example — process report:

```

#!/bin/bash

echo "=== Top 10 Memory Consuming Processes ==="

ps aux --sort=-%mem | awk 'NR<=11 {printf "%-10s %-8s %-8s %s\n", $1, $2, $4, $11}'

echo "=== CPU Usage by User ==="

ps aux | awk 'NR>1 {cpu[$1]+=$3} END {for (user in cpu) print cpu[user]"% ", user}' | sort -rn
```

### 3. sed — Stream Editor

```

sed 's/old/new/'    file    # Replace FIRST occurrence per line

sed 's/old/new/g'   file    # Replace ALL occurrences per line

sed 's/old/new/gi'  file    # Replace all, case insensitive

sed -i 's/old/new/g' file    # Edit file IN PLACE

sed -i.bak 's/old/new/g' f   # In place but keep .bak backup

sed '/pattern/d'     file    # Delete lines matching pattern

sed -n '5,10p'       file    # Print only lines 5 to 10
```

```
sed -n '/error/p' file # Print only lines matching pattern
```

Real sysadmin examples:

```
#!/bin/bash

CONFIG="/etc/nginx/nginx.conf"

# Backup before editing — always!
cp $CONFIG ${CONFIG}.bak

# Change worker processes value
sed -i 's/worker_processes auto/worker_processes 4/g' $CONFIG

# Comment out a line
sed -i 's/^listen 80/#listen 80/' $CONFIG

# Remove all blank lines from a file
sed -i '/^$/d' $CONFIG

# Replace an IP address across a config
OLD_IP="192.168.1.10"
NEW_IP="192.168.1.20"
sed -i "s/$OLD_IP/$NEW_IP/g" $CONFIG

echo "Config updated successfully"
```

■■ Always use `sed -i.bak` in production — never edit config files without a backup.

## 4. cut, sort & uniq

```
# cut — extract specific fields

cut -d: -f1 /etc/passwd # Extract usernames

cut -d: -f1,6 /etc/passwd # Extract username and home dir

cut -d',' -f2 report.csv # Extract second column from CSV

cut -c1-10 file.txt # Extract first 10 characters

# sort

sort file.txt # Sort alphabetically

sort -r file.txt # Reverse sort

sort -n file.txt # Sort numerically
```

```

sort -k2 file.txt                # Sort by second column

sort -t: -k3 -n /etc/passwd     # Sort passwd by UID

# uniq

uniq -c file.txt                # Count occurrences

uniq -d file.txt                # Show only duplicates

# Classic log analysis combo:

cat /var/log/auth.log | grep 'Failed' | awk '{print $11}' | sort | uniq -c | sort -rn

```

■ *sort | uniq -c | sort -rn is the most powerful combo for log analysis — memorise it.*

## 5. Real-World Example — Log Analysis Script

```

#!/bin/bash

# =====

# Script Name: log_analyzer.sh

# Description: Analyzes auth log for security

#             insights and suspicious activity

# Usage      : sudo ./log_analyzer.sh

# =====

LOGFILE="/var/log/auth.log"

REPORT="/tmp/security_report_$(date +%Y%m%d_%H%M%S).txt"

THRESHOLD=10

log() { echo "$1 $2" | tee -a $REPORT; }

if [ ! -f "$LOGFILE" ]; then
    echo "Log file not found: $LOGFILE"; exit 1
fi

echo "==== Security Report - $(date) =====" | tee $REPORT

# Failed login attempts

log "INFO" "=== Failed SSH Login Attempts (Top 10 IPs) ==="

grep "Failed password" $LOGFILE \
    | awk '{print $11}' \

```

```

| sort | uniq -c | sort -rn | head -10 \

| while read COUNT IP; do

    if [ $COUNT -ge $THRESHOLD ]; then

        log "WARNING" "$IP - $COUNT attempts - POSSIBLE BRUTE FORCE"

    else

        log "INFO" "$IP - $COUNT attempts"

    fi

done

# Successful logins

log "INFO" "=== Successful Logins ==="

grep "Accepted password" $LOGFILE \

    | awk '{print $9, "from", $11}' \

    | sort | uniq -c | sort -rn | head -10 | tee -a $REPORT

# Root login attempts

ROOT_ATTEMPTS=$(grep "Failed password for root" $LOGFILE | wc -l)

if [ $ROOT_ATTEMPTS -gt 0 ]; then

    log "WARNING" "$ROOT_ATTEMPTS Failed ROOT login attempts detected!"

fi

echo "==== Report saved to: $REPORT ====="

```

## ■ Exercise

1. Write a script called `disk_report.sh` that:
2. Uses `df` and `awk` to list all mounted filesystems with usage above 20%
3. Uses `grep` and `awk` to extract the top 5 largest files in `/var/log`
4. Uses `sed` to generate a clean formatted report saved to `/tmp/disk_report.txt` replacing any `%` with the word percent
5. Uses `sort` to rank filesystems from highest to lowest usage

## ■ Solution — `disk_report.sh`

```

#!/bin/bash

# =====

```

```

# Script Name: disk_report.sh

# Description: Generates a disk usage report

#           with top large files analysis

# Usage      : ./disk_report.sh

# =====

REPORT="/tmp/disk_report.txt"

THRESHOLD=20

# --- Header ---

echo "==== Disk Usage Report - $(date) =====" | tee $REPORT

echo "" | tee -a $REPORT

# --- Section 1: Filesystems above threshold ---

echo "=== Filesystems Above ${THRESHOLD}% Usage (Ranked) ===" | tee -a $REPORT

df -h | awk 'NR>1 {print $5, $1, $3, $2, $6}' \

    | tr -d '%' \

    | sort -rn \

    | while read USAGE FS USED SIZE MOUNT; do

        if [ "$USAGE" -ge "$THRESHOLD" ]; then

            echo "${USAGE} percent used - $FS mounted on $MOUNT ($USED of $SIZE)" | tee -a $REP

ORT

        fi

    done

echo "" | tee -a $REPORT

# --- Section 2: Top 5 largest files in /var/log ---

echo "=== Top 5 Largest Files in /var/log ===" | tee -a $REPORT

find /var/log -type f -exec du -sh {} + 2>/dev/null \

    | sort -rh | head -5 | awk '{print $1, $2}' | tee -a $REPORT

echo "" | tee -a $REPORT

# --- Clean up % signs in report using sed ---

sed -i 's/%/ percent/g' $REPORT

```

```
echo "===== Report saved to: $REPORT ====="
```

Expected Output:

```
===== Disk Usage Report - 2024-01-15 =====

=== Filesystems Above 20 percent Usage (Ranked) ===

87 percent used - /dev/sda1 mounted on / (42G of 50G)
45 percent used - /dev/sdb1 mounted on /data (90G of 200G)

=== Top 5 Largest Files in /var/log ===

2.1G  /var/log/syslog
845M  /var/log/auth.log
210M  /var/log/kern.log
```

## Module 6 Error Handling & Debugging

This is what separates amateur scripts from production-grade ones. A script that fails silently in production can cause serious damage.

### 1. Exit Codes — The Foundation

```
# Every command returns an exit code. 0=success, non-zero=failure

ls /etc/passwd

echo "Exit code: $?"      # 0 - success

ls /nonexistent

echo "Exit code: $?"      # 2 - failure

# Always check exit codes for critical commands

useradd testuser

if [ $? -ne 0 ]; then

    echo "Failed to create user"

    exit 1

fi

echo "User created successfully"
```

### 2. set Options — Script Safety Switches

```
set -e          # Exit immediately if any command fails

set -u          # Treat unset variables as errors

set -o pipefail # Catch failures inside pipes

set -x          # Debug mode - print every command before executing

# Combine them - use this at top of EVERY production script

set -euo pipefail
```

Why each one matters:

```
#!/bin/bash

set -euo pipefail

# Without set -e, script continues even after rm fails

rm /important/file.txt
```

```
echo "This would still run without set -e - dangerous!"

# Without set -u, typos silently use empty string
echo $USERNMAE    # typo - without set -u this silently returns empty

# Without pipefail, shows exit code 0 even though grep failed
cat /nonexistent | grep 'pattern'

echo $?    # would show 0 without pipefail
```

### 3. trap — Catching Errors and Cleaning Up

```
#!/bin/bash

set -euo pipefail

TMPFILE="/tmp/myscript_$$_temp.txt"

# Cleanup function
cleanup() {
    echo "Cleaning up temporary files..."
    rm -f $TMPFILE
    echo "Cleanup done"
}

# Trap signals
trap cleanup EXIT          # Run cleanup on any exit
trap cleanup ERR           # Run cleanup on error
trap "echo 'Interrupted!'; cleanup; exit 1" INT TERM

# Main script
echo "Working..." > $TMPFILE

sleep 5

echo "Done!"
```

Signal	When it fires
EXIT	Any script exit — normal or error
ERR	Any command returns non-zero exit code

INT	User presses Ctrl+C
TERM	Script receives a kill signal

## 4. Custom Error Functions

```
#!/bin/bash

set -euo pipefail

LOGFILE="/var/log/myscript.log"

log_info() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] [INFO] $1" | tee -a $LOGFILE
}

log_warn() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] [WARN] $1" | tee -a $LOGFILE
}

log_error() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] [ERROR] $1" | tee -a $LOGFILE
    exit 1
}

log_info "Script started"

log_warn "Disk usage is high"

log_error "Failed to connect to database" # This exits the script

log_info "This line will never run"
```

## 5. Debugging with set -x

```
# Debug entire script

#!/bin/bash

set -x

USERNAME="john"

HOME_DIR="/home/$USERNAME"

if [ -d "$HOME_DIR" ]; then
```

```
    echo "Home exists"

fi

# Output with set -x:

# + USERNAME=john

# + HOME_DIR=/home/john

# + '[' -d /home/john ']'

# + echo 'Home exists'


# Debug only a specific section:

echo "Normal section – no debug output"


set -x    # Start debugging

DISK=$(df / | awk 'NR==2 {print $5}')

echo "Disk: $DISK"

set +x    # Stop debugging


echo "Back to normal"
```

## 6. Input Validation Pattern

```
validate_user() {

    local USERNAME=$1

    if [ -z "$USERNAME" ]; then

        log_error "Username cannot be empty"

    fi

    if [[ ! "$USERNAME" =~ ^[a-zA-Z0-9_-]+$ ]]; then

        log_error "Invalid username: only letters, numbers, _ and - allowed"

    fi

    if id "$USERNAME" &>/dev/null; then

        log_error "User $USERNAME already exists"

    fi

    log_info "Username $USERNAME is valid"

}
```

```

validate_ip() {

    local IP=$1

    if [[ ! "$IP" =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]]; then

        log_error "Invalid IP address: $IP"

    fi

    log_info "IP $IP is valid"

}

```

## 7. Real-World — Production-Grade Script with Full Error Handling

```

#!/bin/bash

# =====

# Script Name: safe_backup.sh

# Description: Backs up a directory safely

#               with full error handling

# Usage       : sudo ./safe_backup.sh <source_dir>

# =====

set -euo pipefail

BACKUP_DIR="/backup"

LOGFILE="/var/log/backup.log"

TIMESTAMP=$(date +%Y%m%d_%H%M%S)

TMPFILE="/tmp/backup_$$_${TIMESTAMP}.tar.gz"

log_info() { echo "[$(date '+%F %T')] [INFO] $1" | tee -a $LOGFILE; }

log_error() { echo "[$(date '+%F %T')] [ERROR] $1" | tee -a $LOGFILE; exit 1; }

cleanup() {

    [ -f "$TMPFILE" ] && { log_info "Removing temp file"; rm -f "$TMPFILE"; }

}

trap cleanup EXIT

[ "$EUID" -ne 0 ] && log_error "Must run as root"

[ -z "${1:-}" ] && log_error "Usage: $0 <source_directory>"

[ ! -d "$1" ] && log_error "Source directory not found: $1"

```

```
SOURCE=$1

DEST="$BACKUP_DIR/${basename $SOURCE}_${TIMESTAMP}.tar.gz"

# Disk space check

AVAILABLE=$(df "$BACKUP_DIR" | awk 'NR==2 {print $4}')

SOURCE_SIZE=$(du -sk "$SOURCE" | cut -f1)

if [ "$SOURCE_SIZE" -ge "$AVAILABLE" ]; then

    log_error "Not enough disk space!"

fi

log_info "Starting backup: $SOURCE"

mkdir -p "$BACKUP_DIR"

tar -czf "$TMPFILE" "$SOURCE" || log_error "tar failed"

mv "$TMPFILE" "$DEST" || log_error "Move failed"

log_info "Backup complete: $DEST ($(du -sh $DEST | cut -f1))"
```

## ■ ■ Exercise

1. Write a script called `safe_deploy.sh` using all error handling techniques:
2. Use `set -euo pipefail` at the top
3. Include `log_info`, `log_warn`, `log_error` functions
4. Use `trap cleanup EXIT` to remove temp files
5. Validate all inputs (root check, argument check, directory check)
6. Check disk space before copying files
7. Use `set -x` to debug a specific section of the script

■ **Key Takeaways:** Always start with `set -euo pipefail`. Use `trap cleanup EXIT` always. Validate all inputs. Never assume a command succeeded — always verify with `$?` or `if` statements.

Module 7

Real-World Sysadmin Projects

This is where everything comes together. We'll build 5 complete production-grade scripts that you can actually use in your daily work as a Linux administrator.

#	Script	What it does
1	user_manager.sh	Create, delete, list users with validation & logging
2	health_monitor.sh	Full system health check with alerting & reporting
3	auto_backup.sh	Automated backup with retention & integrity check
4	log_cleanup.sh	Log rotation, compression & cleanup automation
5	server_inventory.sh	Complete server inventory & hardware report

Project 1 — Automated User Management (user\_manager.sh)

```
#!/bin/bash

# =====
# Script Name: user_manager.sh
# Description: Create, delete, and list users
#              with full validation and logging
# Usage       : sudo ./user_manager.sh {create|delete|list} [username]
# =====

set -euo pipefail

LOGFILE="/var/log/user_manager.log"

DEFAULT_SHELL="/bin/bash"

DEFAULT_GROUP="users"

log_info() { echo "[$(date '+%F %T')] [INFO] $1" | tee -a $LOGFILE; }
log_warn() { echo "[$(date '+%F %T')] [WARN] $1" | tee -a $LOGFILE; }
log_error() { echo "[$(date '+%F %T')] [ERROR] $1" | tee -a $LOGFILE; exit 1; }

[ "$EUID" -ne 0 ] && log_error "Must run as root. Use sudo."
```

```
validate_username() {

    local USERNAME=$1

    [ -z "$USERNAME" ] && log_error "Username cannot be empty"

    [[ ! "$USERNAME" =~ ^[a-zA-Z0-9_-]+$ ]] && \

        log_error "Invalid username: only letters, numbers, _ and - allowed"

}

create_user() {

    local USERNAME=$1

    validate_username "$USERNAME"

    if id "$USERNAME" &>/dev/null; then

        log_warn "User $USERNAME already exists - skipping"; return

    fi

    useradd -m -s "$DEFAULT_SHELL" -g "$DEFAULT_GROUP" "$USERNAME" \

        || log_error "Failed to create user: $USERNAME"

    chage -d 0 "$USERNAME"

    echo "$USERNAME:TempPass@123" | chpasswd \

        || log_error "Failed to set password for: $USERNAME"

    log_info "Created: $USERNAME | Home: /home/$USERNAME"

    log_warn "Temporary password set. User must change on first login."

}

delete_user() {

    local USERNAME=$1

    validate_username "$USERNAME"

    ! id "$USERNAME" &>/dev/null && { log_warn "$USERNAME does not exist"; return; }

    ARCHIVE="/backup/${USERNAME}_home_$(date +%Y%m%d).tar.gz"

    mkdir -p /backup

    tar -czf "$ARCHIVE" "/home/$USERNAME" 2>/dev/null \

        && log_info "Home archived to: $ARCHIVE"

    userdel -r "$USERNAME" || log_error "Failed to delete user: $USERNAME"

    log_info "Deleted: $USERNAME | Archive: $ARCHIVE"
```

```

}

list_users() {

    echo ""; echo "==== System Users with Bash Shell ====="

    printf "%-15s %-25s %-10s\n" "USERNAME" "HOME" "LAST LOGIN"

    echo "-----"

    while IFS=: read -r USER _ _ _ HOME SHELL; do

        if [ "$SHELL" = "/bin/bash" ]; then

            LAST=$(lastlog -u "$USER" 2>/dev/null | awk 'NR==2 {print $4,$5,$9}')

            printf "%-15s %-25s %-10s\n" "$USER" "$HOME" "$LAST"

        fi

    done < /etc/passwd

}

ACTION=${1:-}; USERNAME=${2:-}

case "$ACTION" in

    create) create_user "$USERNAME" ;;

    delete) delete_user "$USERNAME" ;;

    list) list_users ;;

    *) echo 'Usage: $0 {create|delete|list} [username]'; exit 1 ;;

esac

```

How to run:

```

sudo ./user_manager.sh create john

sudo ./user_manager.sh delete john

sudo ./user_manager.sh list

```

## Project 2 — System Health Monitor (health\_monitor.sh)

```

#!/bin/bash

# =====

# Script Name: health_monitor.sh

# Description: Full system health check with

#               alerting and detailed reporting

# Usage       : sudo ./health_monitor.sh

```

```
# Schedule      : */5 * * * * /path/to/health_monitor.sh

# =====

set -euo pipefail

LOGFILE="/var/log/health_monitor.log"

REPORT="/tmp/health_report_$(date +%Y%m%d_%H%M%S).txt"

DISK_THRESHOLD=85; MEM_THRESHOLD=90; CPU_THRESHOLD=80

SERVICES=("sshd" "cron" "rsyslog")

ALERT_EMAIL="admin@yourdomain.com"

log_info() { echo "[$(date '+%F %T')] [INFO]  $1" | tee -a $LOGFILE; }

log_warn() { echo "[$(date '+%F %T')] [WARN]  $1" | tee -a $LOGFILE; }

log_crit() { echo "[$(date '+%F %T')] [CRIT]  $1" | tee -a $LOGFILE; }

send_alert() {

    echo "$2" | mail -s "$1" "$ALERT_EMAIL" 2>/dev/null \

    && log_info "Alert sent" || log_warn "Alert email failed"

}

check_disk() {

    log_info "--- Checking Disk Usage ---"

    while read -r USAGE FS MOUNT; do

        if [ "$USAGE" -ge "$DISK_THRESHOLD" ]; then

            log_crit "Disk CRITICAL: $MOUNT at ${USAGE}%"

            send_alert "DISK CRITICAL: $MOUNT" "Disk at ${USAGE}%"

        elif [ "$USAGE" -ge 70 ]; then

            log_warn "Disk WARNING: $MOUNT at ${USAGE}%"

        else

            log_info "Disk OK: $MOUNT at ${USAGE}%"

        fi

    done < <(df -h | awk 'NR>1 {gsub(/%/,"",$5); print $5,$1,$6}')

}

check_memory() {
```

```
log_info "--- Checking Memory Usage ---"

local TOTAL=$(free -m | awk '/Mem:/ {print $2}')

local USED=$(free -m | awk '/Mem:/ {print $3}')

local PERCENT=$(( USED * 100 / TOTAL ))

if [ "$PERCENT" -ge "$MEM_THRESHOLD" ]; then

    log_crit "Memory CRITICAL: ${PERCENT}% used"

    send_alert "MEMORY CRITICAL" "Memory at ${PERCENT}%"

else

    log_info "Memory OK: ${PERCENT}% used"

fi

}

check_services() {

    log_info "--- Checking Services ---"

    for SERVICE in "${SERVICES[@]}; do

        STATUS=$(systemctl is-active "$SERVICE" 2>/dev/null || echo "unknown")

        if [ "$STATUS" = "active" ]; then

            log_info "Service OK: $SERVICE is running"

        else

            log_crit "Service DOWN: $SERVICE - attempting restart"

            systemctl restart "$SERVICE" 2>/dev/null \

                && log_info "$SERVICE restarted OK" \

                || log_crit "Failed to restart $SERVICE"

        fi

    done

}

check_zombies() {

    log_info "--- Checking Zombie Processes ---"

    local ZOMBIES=$(ps aux | awk '$8=="Z"' | wc -l)

    [ "$ZOMBIES" -gt 0 ] \

        && log_warn "Found $ZOMBIES zombie process(es)" \
```

```

        || log_info "No zombie processes"

    }

    log_info "==== Health Monitor Started ====="

    check_disk; check_memory; check_services; check_zombies

    log_info "==== Health Monitor Completed ====="

```

Schedule with cron:

```
*/5 * * * * /opt/scripts/health_monitor.sh
```

## Project 3 — Automated Backup (auto\_backup.sh)

```

#!/bin/bash

# =====
# Script Name: auto_backup.sh
# Description: Automated backup with retention
#              policy and integrity verification
# Usage       : sudo ./auto_backup.sh <source>
# Schedule    : 0 2 * * * /path/to/auto_backup.sh /var/www
# =====

set -euo pipefail

BACKUP_ROOT="/backup"

LOGFILE="/var/log/auto_backup.log"

RETENTION_DAYS=7

TIMESTAMP=$(date +%Y%m%d_%H%M%S)

log_info() { echo "[$(date '+%F %T')] [INFO] $1" | tee -a $LOGFILE; }

log_error() { echo "[$(date '+%F %T')] [ERROR] $1" | tee -a $LOGFILE; exit 1; }

cleanup() { [ -f "${TMPFILE:-}" ] && rm -f "$TMPFILE"; }

trap cleanup EXIT

[ "$EUID" -ne 0 ] && log_error "Must run as root"

[ -z "${1:-}" ] && log_error "Usage: $0 <source_directory>"

[ ! -d "$1" ] && log_error "Source not found: $1"

```

```

SOURCE=$1

SOURCE_NAME=$(basename "$SOURCE")

DEST_DIR="$BACKUP_ROOT/$SOURCE_NAME"

DEST_FILE="$DEST_DIR/${SOURCE_NAME}_${TIMESTAMP}.tar.gz"

TMPFILE="$DEST_DIR/.tmp_${TIMESTAMP}.tar.gz"

CHECKSUM_FILE="${DEST_FILE}.md5"

mkdir -p "$DEST_DIR"

# Disk space check

AVAILABLE=$(df "$BACKUP_ROOT" | awk 'NR==2 {print $4}')

SOURCE_SIZE=$(du -sk "$SOURCE" | cut -f1)

[ "$SOURCE_SIZE" -ge "$AVAILABLE" ] && \

    log_error "Insufficient disk space. Need: ${SOURCE_SIZE}K | Available: ${AVAILABLE}K"

log_info "Starting backup: $SOURCE"

tar -czf "$TMPFILE" -C "$(dirname $SOURCE)" "$SOURCE_NAME" \

    || log_error "Backup failed during compression"

mv "$TMPFILE" "$DEST_FILE"

# Integrity check

md5sum "$DEST_FILE" > "$CHECKSUM_FILE"

md5sum -c "$CHECKSUM_FILE" &>/dev/null \

    && log_info "Integrity check PASSED" \

    || log_error "Integrity check FAILED - backup may be corrupt"

BACKUP_SIZE=$(du -sh "$DEST_FILE" | cut -f1)

log_info "Backup complete | Size: $BACKUP_SIZE | File: $DEST_FILE"

# Retention policy

log_info "Applying retention: keeping last $RETENTION_DAYS days"

find "$DEST_DIR" -name "*.tar.gz" -mtime +$RETENTION_DAYS -exec rm -f {} \;

REMAINING=$(find "$DEST_DIR" -name "*.tar.gz" | wc -l)

log_info "Backups retained: $REMAINING"

```

Schedule with cron:

```
0 2 * * * /opt/scripts/auto_backup.sh /var/www
```

## Project 4 — Log Rotation & Cleanup (log\_cleanup.sh)

```
#!/bin/bash

# =====

# Script Name: log_cleanup.sh

# Description: Rotates, compresses and cleans

#             up old logs automatically

# Usage       : sudo ./log_cleanup.sh

# Schedule    : 0 0 * * 0 /path/to/log_cleanup.sh

# =====

set -euo pipefail

LOG_DIR="/var/log"

ARCHIVE_DIR="/var/log/archive"

LOGFILE="/var/log/log_cleanup.log"

COMPRESS_DAYS=3

DELETE_DAYS=30

MAX_SIZE_MB=100

log_info() { echo "[$(date '+%F %T')] [INFO]  $1" | tee -a $LOGFILE; }

log_warn() { echo "[$(date '+%F %T')] [WARN]  $1" | tee -a $LOGFILE; }

[ "$EUID" -ne 0 ] && { echo "Must run as root"; exit 1; }

mkdir -p "$ARCHIVE_DIR"

log_info "===== Log Cleanup Started ====="

# Compress logs older than N days

log_info "Compressing logs older than $COMPRESS_DAYS days..."

find "$LOG_DIR" -maxdepth 1 -type f -name "*.log" \

    -mtime +$COMPRESS_DAYS ! -name "*.gz" \

    | while read -r LOGF; do

        gzip -f "$LOGF" && log_info "Compressed: $LOGF"
```

```
done

# Move compressed logs to archive

log_info "Moving compressed logs to archive..."

find "$LOG_DIR" -maxdepth 1 -name "*.gz" \

| while read -r GZFILE; do

    mv "$GZFILE" "$ARCHIVE_DIR/" \

    && log_info "Archived: $(basename $GZFILE)"

done

# Delete archives older than retention period

log_info "Deleting archives older than $DELETE_DAYS days..."

find "$ARCHIVE_DIR" -name "*.gz" -mtime +$DELETE_DAYS \

| while read -r OLD; do

    rm -f "$OLD" && log_info "Deleted: $(basename $OLD)"

done

# Truncate oversized active logs

log_info "Checking for oversized active logs..."

find "$LOG_DIR" -maxdepth 1 -type f -name "*.log" \

| while read -r LOGF; do

    SIZE_MB=$(du -m "$LOGF" | cut -f1)

    if [ "$SIZE_MB" -ge "$MAX_SIZE_MB" ]; then

        log_warn "$LOGF is ${SIZE_MB}MB - truncating"

        cp "$LOGF" "${LOGF}.bak"

        : > "$LOGF"

        log_info "Truncated: $LOGF | Backup: ${LOGF}.bak"

    fi

done

ARCHIVE_SIZE=$(du -sh "$ARCHIVE_DIR" | cut -f1)

ARCHIVE_COUNT=$(find "$ARCHIVE_DIR" -name "*.gz" | wc -l)

log_info "Archive size: $ARCHIVE_SIZE | Files: $ARCHIVE_COUNT"
```

```
log_info "==== Log Cleanup Completed ====="
```

Schedule with cron:

```
0 0 * * 0 /opt/scripts/log_cleanup.sh
```

## Project 5 — Server Inventory Report (server\_inventory.sh)

```
#!/bin/bash

# =====

# Script Name: server_inventory.sh

# Description: Generates a full server

#               inventory and hardware report

# Usage       : sudo ./server_inventory.sh

# =====

set -euo pipefail

REPORT="/tmp/inventory_$(hostname)_$(date +%Y%m%d).txt"

section() { echo "" | tee -a $REPORT; echo "==== $1 =====" | tee -a $REPORT; }

info()     { printf "%-20s: %s\n" "$1" "$2" | tee -a $REPORT; }

echo "==== Server Inventory Report =====" | tee $REPORT

echo "Generated: $(date)" | tee -a $REPORT

section "System Information"

info "Hostname"      "$(hostname -f)"

info "OS"            "$(grep PRETTY_NAME /etc/os-release | cut -d= -f2 | tr -d '')"

info "Kernel"        "$(uname -r)"

info "Architecture"  "$(uname -m)"

info "Uptime"        "$(uptime -p)"

info "Last Boot"     "$(who -b | awk '{print $3, $4}')"

section "CPU Information"

info "Model"         "$(grep 'model name' /proc/cpuinfo | head -1 | cut -d: -f2 | xargs)"

info "CPU Cores"     "$(nproc)"

info "Sockets"       "$(grep 'physical id' /proc/cpuinfo | sort -u | wc -l)"
```

```

info "Load Avg"    "${uptime | awk -F'load average:' '{print $2}' | xargs}"

section "Memory Information"

info "Total RAM"   "${free -h | awk '/Mem:/ {print $2}'}"
info "Used RAM"    "${free -h | awk '/Mem:/ {print $3}'}"
info "Free RAM"    "${free -h | awk '/Mem:/ {print $4}'}"
info "Total Swap"  "${free -h | awk '/Swap:/ {print $2}'}"
info "Used Swap"   "${free -h | awk '/Swap:/ {print $3}'}"

section "Disk Information"

df -h | awk 'NR>1 {printf "%-20s %-8s %-8s %-8s %s\n", $1,$2,$3,$4,$5}' | tee -a $REPORT

section "Network Information"

ip -br addr show | tee -a $REPORT

info "Default GW"  "${ip route | awk '/default/ {print $3}'}"

info "DNS Servers" "${grep nameserver /etc/resolv.conf | awk '{print $2}' | tr '\n' ' '}"

section "Active Services"

systemctl list-units --type=service --state=active \

    | awk 'NR>1 && /running/ {print $1}' | head -20 | tee -a $REPORT

section "Currently Logged In Users"

who | tee -a $REPORT

section "Last 5 Logins"

last | head -5 | tee -a $REPORT

echo "" | tee -a $REPORT

echo "==== Report saved to: $REPORT ====="

```

Schedule with cron:

```
0 7 * * 1 /opt/scripts/server_inventory.sh
```

## Scheduling All Scripts with Cron

```
# Edit your crontab

crontab -e
```

```
# Cron syntax:

# MIN  HOUR  DAY  MONTH  WEEKDAY  COMMAND

#  *    *    *    *    *    *

# Run health monitor every 5 minutes

*/5 * * * * /opt/scripts/health_monitor.sh

# Run backup daily at 2 AM

0 2 * * * /opt/scripts/auto_backup.sh /var/www

# Run log cleanup every Sunday at midnight

0 0 * * 0 /opt/scripts/log_cleanup.sh

# Run inventory report every Monday at 7 AM

0 7 * * 1 /opt/scripts/server_inventory.sh

# Run user audit daily at 6 AM

0 6 * * * /opt/scripts/user_audit.sh
```

Module	Topic	Key Skills Gained
1	Script Foundations	Shebang, variables, arguments, \$() command capture, exit codes
2	Conditionals & Logic	if/elif/else, case, file/string/numeric test operators
3	Loops	for, while, until, arrays, break/continue, line-by-line reading
4	Functions	Reusable code, local vars, return values, logger function
5	Files & Text	grep, awk, sed, cut, sort, uniq — log analysis pipelines
6	Error Handling	set -euo pipefail, trap, set -x debugging, input validation
7	Real-World Projects	user_manager, health_monitor, auto_backup, log_cleanup, inventory

■ Bash Scripting Module Complete!

You have mastered Bash scripting from foundations to production-grade sysadmin automation.

Next up: [Python for Linux Administrators](#) →