■ Linux Administrator

From Basics to Production-Grade Scripts

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. 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   |   Version: 1.0
# ============================================

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}')

echo "===== System Information ====="
echo "Hostname  : $HOSTNAME"
echo "OS        : $OS"
echo "Uptime    : $UPTIME"
echo "Disk Used : $DISK"
echo "Memory    : $MEMORY"
echo "=============================="
```

3. Variables

```
NAME="LinuxAdmin"           # No spaces around =

CURRENT_USER=$(whoami)      # Capture command output

echo "Running as: $CURRENT_USER"

readonly MAX_RETRIES=3      # Read-only constant
```

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

4. Special Built-in Variables

```
$0   # Script name          $1, $2  # Arguments

$#   # Total argument count  $@      # All arguments

$?   # Last exit code        $$      # Script PID

$USER # Current user
```

5. Making a Script Executable

```
chmod +x system_info.sh    # Give execute permission

./system_info.sh           # Run it

sudo ./system_info.sh      # Run as root
```

■ *Always use UPPERCASE for variable names so they stand out clearly in your scripts.*

Module 2

Conditionals & Logic

1. if / elif / else Structure

```
if [ condition ]; then

    # do something

elif [ another_condition ]; then

    # do something else

else

    # fallback

fi
```

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

2. File & Directory Test Operators

```
[ -f /etc/passwd ]      # File EXISTS

[ -d /var/log ]         # DIRECTORY exists

[ -r /etc/shadow ]      # File is READABLE

[ -w /tmp/test ]        # File is WRITABLE

[ -x /usr/bin/python ]  # File is EXECUTABLE

[ -s /var/log/syslog ]  # File is NOT EMPTY

[ -e /etc/hosts ]       # Path EXISTS (file or dir)
```

3. String & Numeric Operators

```
# Strings

[ -z "$VAR" ]    # Empty         [ -n "$VAR" ]  # Not empty

[ "$A" = "$B" ]  # Equal         [ "$A" != "$B" ] # Not equal


# Numbers

[ $A -eq $B ]    # Equal         [ $A -ne $B ]  # Not equal

[ $A -gt $B ]    # Greater than  [ $A -lt $B ]  # Less than
[ $A -ge $B ]    # >= (gte)      [ $A -le $B ]  # <= (lte)
```

4. Real Example — Service Check

```
#!/bin/bash

SERVICE=$1

STATUS=$(systemctl is-active $SERVICE)


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

    echo "Service $SERVICE is running"

else

    echo "Not running — restarting..."

    systemctl restart $SERVICE

fi
```

5. Combining Conditions & case Statement

```
# AND / OR

[ $DISK -gt 80 ] && [ -f "/etc/alert.conf" ] && echo "Alert!"

[ "$USER" = "root" ] || [ "$USER" = "admin" ] && echo "Privileged"


# case statement

case "$ACTION" in

    start)   systemctl start nginx  ;;

    stop)    systemctl stop nginx   ;;

    restart) systemctl restart nginx ;;

    *) echo 'Usage: {start|stop|restart}'; exit 1 ;;

esac
```

6. Check if Running as Root

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

    echo "Must be run as root. Use sudo."
    exit 1

fi
```

🟩 *Put the root check at the very top of every script that requires elevated privileges.*

Module 3 Loops

1. for Loop

```
# Loop over a list
for SERVICE in nginx sshd cron firewalld; 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
```

2. while Loop — Retry Logic

```
MAX_RETRIES=3; COUNT=0
while [ $COUNT -lt $MAX_RETRIES ]; do
    STATUS=$(systemctl is-active nginx)
    if [ "$STATUS" = "active" ]; then
        echo "Running!"; break
    fi
    COUNT=$((COUNT + 1))
    echo "Attempt $COUNT — retrying in 5s..."
    sleep 5; systemctl restart nginx
done
```

3. Reading a File Line by Line

```
while IFS= read -r SERVER; do
```

```
    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 the most common loop pattern in real sysadmin scripting.

4. until Loop — Wait for Service

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

    echo "Waiting for nginx to start..."

    sleep 3

done

echo "nginx is now active!"
```

5. Loop Control & Arrays

```
# break exits loop entirely, continue skips current iteration

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

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

    [ "$SERVER" = "db01" ] && continue  # skip db01

    echo "Processing: $SERVER"

done


# C-style for loop

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

    echo "Iteration: $i"

done
```

Module 4 Functions

1. Basic Function Syntax

```
# Define
greet_user() {
    echo "Hello, $1!"
}

# Call
greet_user "John"
```

Always define functions BEFORE calling them. Bash reads top to bottom.

2. Return Values & Exit Codes

```
is_service_running() {
    [ "$(systemctl is-active $1)" = "active" ]
    return $?   # 0=success 1=failure
}

if is_service_running nginx; then
    echo "nginx is UP"
else
    echo "nginx is DOWN"
fi

# Return text — use echo + $()
get_disk_usage() {
    df / | awk 'NR==2 {print $5}' | tr -d '%'
}
USAGE=$(get_disk_usage)
```

3. Local Variables — Always Use Them

```
check_disk() {
    local THRESHOLD=$1      # local prevents global overwrite
    local USAGE=$(df / | awk 'NR==2 {print $5}' | tr -d '%')
    [ $USAGE -ge $THRESHOLD ] && return 1 || return 0
}
```

4. The Logger Function — Production Essential

```
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 "Service failed — exiting"
```

🟩 *Include log_info, log_warn, and log_error in EVERY production script you write.*

Module 5

Working with Files & Text

1. grep — Searching Text

```
grep -i  'error'  /var/log/syslog      # Case insensitive

grep -r  'failed' /var/log/            # Recursive

grep -v  'INFO'   /var/log/app.log     # Invert (exclude match)

grep -n  'error'  /var/log/app.log     # Show line numbers

grep -c  'error'  /var/log/app.log     # Count matching lines

grep -A3 'error'  /var/log/app.log     # 3 lines AFTER match

grep -B3 'error'  /var/log/app.log     # 3 lines BEFORE match

grep -E  'error|failed|critical' file  # Extended regex
```

2. awk — Column & Field Processing

```
# Key variables: $0=full line  $1=field1  $NF=last field  NR=line#

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

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

awk 'NR>1 && $5+0 > 80 {print $1}' df_output      # Conditional

ps aux | awk '{sum+=$6} END {print sum/1024,"MB"}'  # Sum column
```

3. sed — Stream Editor

```
sed 's/old/new/'    file     # Replace FIRST occurrence

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

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

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

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

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

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

4. cut, sort & uniq

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

cut -d: -f1,6 /etc/passwd        # Username + home dir


sort -r file.txt                 # Reverse sort

sort -n file.txt                 # Numeric sort

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


uniq -c file.txt                 # Count occurrences

# Classic log analysis combo:

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

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

Module 6 Error Handling & Debugging

1. 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


# Combine — put this at top of EVERY production script

set -euo pipefail
```

2. trap — Catching Errors & Cleaning Up

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


cleanup() {

    echo "Cleaning up..."

    rm -f $TMPFILE

}


trap cleanup EXIT          # Run on any exit

trap cleanup ERR           # Run on error

trap 'cleanup; exit 1' INT TERM  # Ctrl+C or kill
```

Signal	When it fires
EXIT	Any script exit — normal or error
ERR	Any command returns non-zero
INT	User presses Ctrl+C
TERM	Script receives kill signal

3. Debugging with set -x

```
# Debug specific section only
echo 'Normal section — no debug'

set -x    # Start debugging
DISK=$(df / | awk 'NR==2 {print $5}')
echo "Disk: $DISK"
set +x    # Stop debugging

echo 'Back to normal'
```

4. Input Validation Pattern

```
validate_username() {
    local USERNAME=$1
    [ -z "$USERNAME" ] && log_error "Username empty"
    [[ ! "$USERNAME" =~ ^[a-zA-Z0-9_-]+$ ]] && \
        log_error "Invalid chars in username"
    id "$USERNAME" &>/dev/null && \
        log_error "User already exists"
}
```

🟩 *Always validate inputs before doing anything destructive — check empty, type, and existence.*

Module 7 Real-World Sysadmin Projects

This module contains 5 complete production-grade scripts that tie everything together. Each script demonstrates real patterns used in professional Linux environments.

#	Script	Description
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 — User Manager (user_manager.sh)

Create, delete, and list system users with full validation, home directory archiving on deletion, and forced password change on first login.

```
#!/bin/bash
set -euo pipefail
LOGFILE='/var/log/user_manager.log'
DEFAULT_SHELL='/bin/bash'

log_info()  { echo "[$(date '+%F %T')] [INFO]  $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'

create_user() {
    local USERNAME=$1
    id "$USERNAME" &>/dev/null && { log_info "$USERNAME exists"; return; }
    useradd -m -s "$DEFAULT_SHELL" "$USERNAME" || log_error 'useradd failed'
    chage -d 0 "$USERNAME"                       # Force pw change
    echo "$USERNAME:TempPass@123" | chpasswd
    log_info "Created: $USERNAME"
}

delete_user() {
    local USERNAME=$1
    ARCHIVE="/backup/${USERNAME}_$(date +%Y%m%d).tar.gz"
    mkdir -p /backup
    tar -czf "$ARCHIVE" "/home/$USERNAME" 2>/dev/null
    userdel -r "$USERNAME" && log_info "Deleted: $USERNAME | Archive: $ARCHIVE"
}

case "${1:-}" in
    create) create_user "${2:-}" ;;
    delete) delete_user "${2:-}" ;;
    *) echo 'Usage: $0 {create|delete} username'; exit 1 ;;
esac
```

Usage: sudo ./user_manager.sh create john | sudo ./user_manager.sh delete john

Project 2 — Health Monitor (health_monitor.sh)

Comprehensive system health check covering disk, memory, CPU, services, and zombie processes. Sends email alerts on critical thresholds and saves a summary report.

```
#!/bin/bash
set -euo pipefail
DISK_THRESHOLD=85; MEM_THRESHOLD=90; CPU_THRESHOLD=80
SERVICES=('sshd' 'cron' 'rsyslog')

check_disk() {
    while read -r USAGE FS MOUNT; do
        [ "$USAGE" -ge "$DISK_THRESHOLD" ] && \
            log_crit "DISK: $MOUNT at ${USAGE}%" || \
            log_info "Disk OK: $MOUNT at ${USAGE}%"
    done < <(df -h | awk 'NR>1 {gsub(/%/,"",$5); print $5,$1,$6}')
}

check_memory() {
    local TOTAL=$(free -m | awk '/Mem:/ {print $2}')
    local USED=$(free -m  | awk '/Mem:/ {print $3}')
    local PCT=$(( USED * 100 / TOTAL ))
    [ "$PCT" -ge "$MEM_THRESHOLD" ] && \
        log_crit "Memory CRITICAL: ${PCT}%" || \
        log_info "Memory OK: ${PCT}%"
}

check_services() {
    for SVC in "${SERVICES[@]}"; do
        systemctl is-active "$SVC" &>/dev/null \
            && log_info "OK: $SVC" \
            || { log_crit "DOWN: $SVC"; systemctl restart "$SVC"; }
    done
}

check_disk; check_memory; check_services
```

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

Project 3 — Auto Backup (auto_backup.sh)

Backs up a directory with disk space pre-check, md5sum integrity verification, automatic retention policy, and safe temp file handling via trap.

```
#!/bin/bash
set -euo pipefail
BACKUP_ROOT='/backup'; RETENTION_DAYS=7
```

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

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_dir>'
[ ! -d "$1" ]     && log_error "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"

mkdir -p "$DEST_DIR"
tar -czf "$TMPFILE" -C "$(dirname $SOURCE)" "$SOURCE_NAME"
mv "$TMPFILE" "$DEST_FILE"

# Integrity check
md5sum "$DEST_FILE" > "${DEST_FILE}.md5"
md5sum -c "${DEST_FILE}.md5" &>/dev/null \
    && log_info 'Integrity OK' || log_error 'Integrity FAILED'

# Retention policy
find "$DEST_DIR" -name '*.tar.gz' -mtime +$RETENTION_DAYS -delete
log_info "Backup complete: $DEST_FILE"
```

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

Project 4 — Log Cleanup (log_cleanup.sh)

Automatically compresses logs older than N days, archives them, deletes old archives past the retention period, and truncates oversized active log files.

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

```
set -euo pipefail

LOG_DIR='/var/log'; ARCHIVE_DIR='/var/log/archive'

COMPRESS_DAYS=3; DELETE_DAYS=30; MAX_SIZE_MB=100


mkdir -p "$ARCHIVE_DIR"


# Compress logs older than N days

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

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

    | while read -r LOGF; do

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

    done


# Archive compressed logs

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

    -exec mv {} "$ARCHIVE_DIR/" \;


# Delete archives past retention

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


# Truncate oversized active logs

find "$LOG_DIR" -maxdepth 1 -type f -name '*.log' | while read -r F; do

    SIZE=$(du -m "$F" | cut -f1)

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

        cp "$F" "${F}.bak"; : > "$F"

        log_warn "Truncated: $F (was ${SIZE}MB)"

    fi

done
```

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

Project 5 — Server Inventory (server_inventory.sh)

Generates a complete server inventory report covering system info, CPU, memory, disk, network, active services, and recent logins. Saves to a timestamped file.

```bash
#!/bin/bash
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

section 'System Information'
info 'Hostname'  "$(hostname -f)"
info 'OS'        "$(grep PRETTY_NAME /etc/os-release | cut -d= -f2 | tr -d '"')"
info 'Kernel'    "$(uname -r)"
info 'Uptime'    "$(uptime -p)"

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

section 'Memory Information'
info 'Total RAM' "$(free -h | awk '/Mem:/ {print $2}')"
info 'Used RAM'  "$(free -h | awk '/Mem:/ {print $3}')"

section 'Disk Information'
df -h | awk 'NR>1 {print $1,$2,$3,$4,$5}' | tee -a $REPORT

section 'Network Information'
ip -br addr show | tee -a $REPORT
```

```
section 'Active Services'

systemctl list-units --type=service --state=active | head -20 | tee -a $REPORT
```

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

Cron Syntax

```
# MIN  HOUR  DAY  MONTH  WEEKDAY  COMMAND

#  *    *     *    *       *


*/5 * * * *  /opt/scripts/health_monitor.sh     # Every 5 mins

0   2 * * *  /opt/scripts/auto_backup.sh /var/www # Daily 2 AM

0   0 * * 0  /opt/scripts/log_cleanup.sh         # Every Sunday

0   7 * * 1  /opt/scripts/server_inventory.sh    # Every Monday 7 AM

0   6 * * *  /opt/scripts/user_audit.sh          # Daily 6 AM
```

Module	Topic	Key Skills Gained
1	Script Foundations	Shebang, variables, arguments, $? exit codes
2	Conditionals & Logic	if/elif/else, case, file/string/numeric operators
3	Loops	for, while, until, arrays, break/continue
4	Functions	Reusable code, local vars, return values, logging
5	Files & Text	grep, awk, sed, cut, sort, uniq pipelines
6	Error Handling	set -euo pipefail, trap, set -x debugging
7	Real-World Projects	5 production scripts: users, health, backup, logs, inventory

You have completed the Bash Scripting module and are ready to move on to Python!

Next up: Python for Linux Administrators