BASH Command & Shell Scripting Mastery for DevOps Professionals

Easy: Foundational Command Fluency

1. Navigation & Inspection

Task: List all files, including hidden, in long format sorted by modification time (newest first).

Solution:

```
ls -laht
```

- `-l`: Long listing
- `-a`: All (including hidden files)
- `-h`: Human-readable file sizes
- `-t`: Sort by modification time

Alternative:

```
ls -larth  # Sorts oldest first
```

2. File Creation

Task: Create `project/{dev,prod,test}/config` with a single command.

Solution:

```
mkdir -p project/{dev,prod,test}/config
```

- `-p`: Creates parent directories as needed
- `{a,b,c}`: Brace expansion

3. Basic File Operations

Task: Copy all `.log` files from `/var/log` to `~/backups/logs`, preserving attributes.

Solution:

```
cp -a /var/log/*.log ~/backups/logs/
```

- `-a`: Archive mode (preserves all attributes)

Alternative:

```
rsync -a /var/log/*.log ~/backups/logs/  # Better for large files
```

4. User & Permissions

Task: Change owner of `deploy.sh` to user `deploy` and group `www-data`. Give owner execute and group read.

Solution:

```
sudo chown deploy:www-data deploy.sh
chmod u+x,g+r deploy.sh
```

Alternative (Numeric):

```
chmod 740 deploy.sh  # 7=rwx for owner, 4=r for group, 0=no others
```

5. Text Manipulation (head/tail)

Task: Display first 5 and last 10 lines of `application.log`.

Solution:

```
(head -5 application.log; tail -10 application.log)
```

Alternative:

```
tail -10 application.log | cat <(head -5 application.log) -
```

6. Searching (grep)

Task: Find lines with `"500"` in `nginx.access.log` and save to `errors.log`.

Solution:

```
grep "500" nginx.access.log > errors.log
```

Alternative:

```
grep -c "500" nginx.access.log  # Count occurrences
```

7. Process Management

Task: Find PID of `nginx` and send it `SIGHUP`.

Solution:

```
sudo kill -HUP $(pgrep nginx)
```

Alternative:

```
sudo pkill -HUP nginx  # Directly by process name
```

8. Disk Usage

Task: Find top 5 largest files/directories in `/home`.

Solution:

```
du -ah /home | sort -rh | head -5
```

Alternative:

```
ncdu /home  # Interactive tool
```

9. Downloading

Task: Use `curl` to download a file and save with different name.

Solution:

```
curl -o my_new_file.zip https://example.com/old_file.zip
```

Alternative:

```
wget -O my_new_file.zip https://example.com/old_file.zip
```

10. Basic Piping

Task: List running processes, filter for user, and count.

Solution:

```
ps aux | grep "$USER" | grep -v grep | wc -l
```

Alternative:

```
pgrep -u $USER | wc -l  # More reliable
```

Medium: Scripting Fundamentals & System Interaction

11. Simple Script Skeleton

Task: Script that checks if a file exists.

Solution (**`check_file.sh`**):

```
#!/bin/bash
if [[ -f "$1" ]]; then
    echo "File exists."
else
    echo "File does not exist."
fi
```

12. For Loop

Task: Ping hosts from `hosts.txt` and report reachability.

Solution (**`ping_hosts.sh`**):

```
#!/bin/bash
for host in $(cat hosts.txt); do
    if ping -c 1 -W 1 "$host" &> /dev/null; then
        echo "$host is reachable."
    else
        echo "$host is unreachable."
    fi
done
```

13. User Input

Task: Ask for user's name and greet them.

Solution (**`greeter.sh`**):

```
#!/bin/bash
read -p "What is your name? " name
echo "Hello, $name!"
```

14. Argument Parsing

Task: Script with `-d` for directory and `-e` for extension to count files.

Solution (**`count_files.sh`**):

```
#!/bin/bash
while getopts "d:e:" opt; do
    case $opt in
        d) target_dir="$OPTARG" ;;
        e) extension="$OPTARG" ;;
        *) echo "Invalid option"; exit 1 ;;
    esac
done
find "$target_dir" -maxdepth 1 -name "*$extension" -type f | wc -l
```

15. System Info Script

Task: Output a system report.

Solution (**`sysreport.sh`**):

```
#!/bin/bash
echo "=== System Report ==="
echo "Date: $(date)"
```

```
echo "Disk Usage:"
df -h / | awk 'NR==2{print $5}'
echo "Memory Usage:"
free -h | awk '/Mem:/{print $3 "/" $2}'
echo "Top 5 CPU Processes:"
ps -eo pid,comm,%cpu --sort=-%cpu | head -n 6
```

16. Backup Script

Task: Create a timestamped backup.

Solution (**backup.sh**):

```
#!/bin/bash
source_dir="$1"
backup_dir="/backups"
timestamp=$(date +%Y%m%d_%H%M%S)
backup_name="backup_$(basename "$source_dir")_$timestamp.tar.gz"
tar -czf "$backup_dir/$backup_name" -C "$(dirname "$source_dir")" "$(basename "$source_dir")
echo "Backup created: $backup_dir/$backup_name"
```

17. Log Analyzer

Task: Count HTTP status codes in a log file.

Solution (**log_analyzer.sh**):

```
#!/bin/bash
log_file="$1"
echo "HTTP Status Code Counts:"
grep -oE '" [0-9]{3} ' "$log_file" | sort | uniq -c | sort -nr
```

18. String Manipulation

Task: Extract filename and directory from a full path.

Solution (**path_parser.sh**):

```
#!/bin/bash
full_path="$1"
filename=$(basename "$full_path")
dirname=$(dirname "$full_path")
echo "Filename (without path): ${filename%.*}"
echo "Directory: $dirname"
```

19. API Interaction

Task: Use `curl` and `jq` to parse a JSON API response.

Solution (**github_user.sh**):

```
#!/bin/bash
username="$1"
response=$(curl -s "https://api.github.com/users/$username")
login=$(echo "$response" | jq -r '.login')
name=$(echo "$response" | jq -r '.name')
echo "Username: $login"
echo "Name: $name"
```

20. Cron Job

Task: Schedule backup script to run weekly.

Solution:

```
# Add to crontab (crontab -e)
0 2 * * 0 /path/to/backup.sh /path/to/source_dir >> /var/log/backup.log 2>&1
```

21. Function

Task: Create a `log_message` function.

Solution (**logger.sh**):

```
#!/bin/bash
LOGFILE="./script.log"
log_message() {
    echo "$(date '+%Y-%m-%d %H:%M:%S') - $1" >> "$LOGFILE"
}
log_message "Script started"
# ... do work ...
log_message "Script finished"
```

22. Check for Tools

Task: Check if `docker`, `git`, and `jq` are installed.

Solution (**check_deps.sh**):

```
#!/bin/bash
exit_code=0
for cmd in docker git jq; do
    if ! command -v "$cmd" &> /dev/null; then
        echo "Error: $cmd is not installed." >&2
        exit_code=1
    fi
done
if [[ $exit_code -ne 0 ]]; then
```

```
    exit $exit_code
fi
echo "All dependencies are met."
```

23. File Modification

Task: Comment out lines containing `DEBUG` in `.conf` files.

Solution (**`comment_debug.sh`**):

```
#!/bin/bash
find /path/to/configs -name "*.conf" -exec sed -i '/DEBUG/s/^/#/' {} \;
```

24. Interactive Delete

Task: Find files >30 days old and delete interactively.

Solution (**`clean_old.sh`**):

```
#!/bin/bash
find /path/to/dir -type f -mtime +30 -exec rm -i {} \;
```

25. Port Check

Task: Check if a remote port is open.

Solution (**`check_port.sh`**):

```
#!/bin/bash
host="$1"
port="$2"
timeout 1 bash -c "cat < /dev/null > /dev/tcp/$host/$port" 2>/dev/null
if [[ $? -eq 0 ]]; then
    echo "Port $port on $host is OPEN."
else
    echo "Port $port on $host is CLOSED."
fi
```

Hard: Advanced Scripting & Robustness

26. Error Handling

Task: Make backup script robust with error handling.

Solution (**`robust_backup.sh`**):

```
#!/bin/bash
set -euo pipefail
source_dir="${1:-}"
backup_dir="/backups"
```

```
cleanup() {
    echo "Cleaning up on exit..."
    rm -f "$backup_dir/${backup_name:-NOTSET}"
}

trap cleanup EXIT ERR INT TERM

[[ -z "$source_dir" ]] && { echo "Usage: $0 <source_dir>"; exit 1; }
[[ ! -d "$source_dir" ]] && { echo "Error: $source_dir does not exist."; exit 1; }
[[ ! -w "$backup_dir" ]] && { echo "Error: $backup_dir is not writable."; exit 1; }

timestamp=$(date +%Y%m%d_%H%M%S)
backup_name="backup_$(basename "$source_dir")_$timestamp.tar.gz"

if ! tar -czf "$backup_dir/$backup_name" -C "$(dirname "$source_dir")" "$(basename "$source_
    echo "Error: tar command failed." >&2
    exit 1
fi

trap - EXIT ERR INT TERM
echo "Backup successful: $backup_dir/$backup_name"
```

27. Configuration Parsing

Task: Parse a simple key=value config file.

Solution (**load_config.sh**):

```
#!/bin/bash
config_file="${1:-config.conf}"
while IFS= read -r line; do
    cleaned_line="${line%%#*}"
    cleaned_line="${cleaned_line##*( )}"
    cleaned_line="${cleaned_line%%*( )}"
    if [[ -n "$cleaned_line" && "$cleaned_line" == *=* ]]; then
        key="${cleaned_line%%=*}"
        value="${cleaned_line#*=}"
        declare -- "$key"="$value"
        echo "Config: $key=$value"
    fi
done < <(grep -v '^#\|^$' "$config_file")
echo "Loaded value for DB_HOST: ${DB_HOST:-Not Set}"
```

28. JSON/CSV Processing

Task: Process a CSV and create users.

Solution (`create_users.sh`):

```
#!/bin/bash
input_file="users.csv"
while IFS=',' read -r username uid group; do
    if [[ -z "$username" || "$username" =~ [^a-zA-Z0-9_] ]]; then
        echo "Skipping invalid username: $username" >&2
        continue
    fi
    echo "Creating user: $username with UID:$uid and group:$group"
    # sudo useradd -u "$uid" -G "$group" "$username"
done < <(tail -n +2 "$input_file")
```

29. Interactive Menu

Task: Create a text-based menu.

Solution (`menu.sh`):

```
#!/bin/bash
while true; do
    echo "1. Check Disk Usage"
    echo "2. Check Memory Usage"
    echo "3. Exit"
    read -p "Please select an option [1-3]: " choice
    case $choice in
        1) df -h ;;
        2) free -h ;;
        3) echo "Exiting..."; exit 0 ;;
        *) echo "Invalid option. Please try again." ;;
    esac
    echo
done
```

30. Log Monitoring

Task: Tail a log and alert on repeated errors.

Solution (`log_monitor.sh`):

```
#!/bin/bash
log_file="$1"
error_pattern="ERROR"
threshold=5
interval=60
echo "Monitoring $log_file for '$error_pattern'..."
tail -f "$log_file" | while read -r line; do
    if echo "$line" | grep -q "$error_pattern"; then
```

```
        count=$((count + 1))
        echo "Error detected. Count: $count"
        if (( count >= threshold )); then
            echo "ALERT: $threshold errors detected in the last minute!"
            count=0
        fi
        (sleep $interval && count=$((count - 1))) &
    fi
done
```

31. Git Automation

Task: Automate git add, commit, push.

Solution (**git_auto.sh**):

```
#!/bin/bash
if [[ -z $(git status --porcelain) ]]; then
    echo "No changes to commit."
    exit 0
fi
git add .
commit_message="Auto-commit: $(date '+%Y-%m-%d %H:%M:%S')"
git commit -m "$commit_message"
current_branch=$(git symbolic-ref --short HEAD)
git push origin "$current_branch"
```

32. Docker Container Manager

Task: List, stop, and remove containers.

Solution (**docker_manager.sh**):

```
#!/bin/bash
echo "Running containers:"
docker ps --format "table {{.ID}}\t{{.Names}}"
read -p "Enter container name or ID to stop (or 'quit'): " container_input
if [[ "$container_input" == "quit" ]]; then exit 0; fi
echo "Stopping container $container_input..."
docker stop "$container_input"
read -p "Remove container $container_input? (y/N): " confirm_remove
if [[ "$confirm_remove" == "y" ]]; then
    docker rm "$container_input"
    echo "Container removed."
fi
```

33. Password Generator

Task: Generate a random, secure password.

Solution (**gen_passwd.sh**):

```
#!/bin/bash
length=${1:-16}
upper="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
lower="abcdefghijklmnopqrstuvwxyz"
numbers="0123456789"
symbols='!@#$%^&*()_+-={}[]|:;<>,.?/'
all_chars="${upper}${lower}${numbers}${symbols}"
password=$(head /dev/urandom | tr -dc "$all_chars" | head -c "$length")
echo "Generated Password: $password"
```

34. Parallel Execution

Task: Ping 50 hosts concurrently.

Solution with **parallel**:

```
cat hosts.txt | parallel -j 20 'ping -c 1 -W 1 {} &> /dev/null && echo {} is UP || echo {} i
```

Solution with background processes:

```
while IFS= read -r host; do
    ( ping -c 1 -W 1 "$host" &> /dev/null && echo "$host is UP" ) &
done < hosts.txt
wait
```

35. Self-Documenting Script

Task: Script with `getopts` for help, verbose, file.

Solution (**professional_script.sh**):

```
#!/bin/bash
usage() {
    echo "Usage: $0 [-v] [-h] -f <config_file>" >&2
    echo "  -v: Enable verbose mode."
    echo "  -h: Show this help message."
    echo "  -f: Specify configuration file."
}

VERBOSE=false
CONFIG_FILE=""

while getopts "vhf:" opt; do
    case $opt in
```

```
        v) VERBOSE=true ;;
        h) usage; exit 0 ;;
        f) CONFIG_FILE="$OPTARG" ;;
        *) usage; exit 1 ;;
    esac
done

if [[ -z "$CONFIG_FILE" ]]; then
    echo "Error: -f option is required." >&2
    usage
    exit 1
fi

[[ "$VERBOSE" == "true" ]] && echo "Verbose mode enabled. Using config: $CONFIG_FILE"
```

Tricky: Edge Cases & Deep Understanding

36. Safe `rm`

Task: Create a safe `rm` wrapper.

Solution:

```
safe_rm() {
    local trash_dir="${HOME}/.trash"
    mkdir -p "$trash_dir"
    for item in "$@"; do
        mv -- "$item" "$trash_dir/$(basename "$item")_$(date +%s)"
    done
    echo "Moved $# item(s) to $trash_dir"
}
# alias rm="safe_rm"  # Use with caution
```

37. String Replacement in Bulk

Task: Replace `old.example.com` with `new.example.com` in `.php` and `.html` files.

Solution:

```
find /project -type f \( -name "*.php" -o -name "*.html" \) -exec sed -i 's/old\.example\.co
```

38. Validate IP Address

Task: Function to validate IPv4 address.

Solution:

```
validate_ip() {
    local ip=$1
```

```
    local stat=1
    if [[ $ip =~ ^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$ ]]; then
        IFS='.' read -r -a octets <<< "$ip"
        for octet in "${octets[@]}"; do
            if (( octet > 255 )); then
                return 1
            fi
        done
        stat=0
    fi
    return $stat
}
```

39. Script Locking

Task: Ensure only one instance runs.

Solution (**singleton.sh**):

```
#!/bin/bash
LOCKFILE="/tmp/$(basename "$0").lock"
if [[ -e "$LOCKFILE" ]]; then
    pid=$(cat "$LOCKFILE")
    if kill -0 "$pid" 2>/dev/null; then
        echo "Script is already running (PID: $pid). Exiting." >&2
        exit 1
    else
        echo "Removing stale lock file." >&2
        rm -f "$LOCKFILE"
    fi
fi
echo $$ > "$LOCKFILE"
trap 'rm -f "$LOCKFILE"; exit' EXIT ERR INT TERM
```

40. SSH Automation

Task: Run `df -h` on multiple servers.

Solution (**ssh_multi_run.sh**):

```
#!/bin/bash
while IFS= read -r server; do
    echo "=== Disk Usage on $server ==="
    ssh -o ConnectTimeout=5 -o BatchMode=yes "$server" "df -h /" 2>/dev/null || echo "Failed
    echo
done < servers.txt
```

41. The Space Problem

Task: Loop over files handling special characters.

Solution:

```
find . -maxdepth 1 -type f -print0 | while IFS= read -r -d '' file; do
    echo "Processing: $file"
done
```

42. Quoting Hell

Task: Demonstrate `$@` vs `$*`.

Solution:

```
# test_args.sh
echo "Number of args: $#"
echo "Using \$@ (with quotes):"
for arg in "$@"; do echo "[$arg]"; done
echo "Using \$* (with quotes):"
for arg in "$*"; do echo "[$arg]"; done
```

43. Source vs Execute

Task: Demonstrate sourcing vs executing.

Solution:

```
# script.sh: my_var="Hello from script"
./script.sh    # my_var not available after
source script.sh # my_var available in current shell
```

44. Subshell Gotcha

Task: Show variable scope issue.

Solution:

```
count=0
echo -e "line1\nline2" | while read -r line; do
    ((count++))
done
echo "Count from subshell: $count" # 0

count=0
while read -r line; do
    ((count++))
done < <(echo -e "line1\nline2")
echo "Count without subshell: $count" # 2
```

45. Exit Code Chaining

Task: Chain commands based on exit codes.

Solution:

```
command1 && command2  # Run command2 only if command1 succeeds
command1 || command2  # Run command2 only if command1 fails
```

46. The Null Command

Task: Demonstrate the colon : command.

Solution:

```
:  # Does nothing, successfully

# Placeholder
if some_condition; then
    : # TODO: implement later
else
    do_something
fi

# Infinite loop
while :
do
    sleep 1
done
```

47. Parameter Expansion Magic

Task: Demonstrate parameter expansions.

Solution:

```
var="hello.world.txt"
echo "Length: ${#var}"           # 14
echo "Remove suffix: ${var%.*}"  # hello.world
echo "Remove prefix: ${var##*.}" # txt
unset maybe_empty
echo "Default if unset: ${maybe_empty:-default_value}" # default_value
```

48. Arithmetic in Bash

Task: Calculate average using bash.

Solution:

```
sum=0
count=0
while IFS= read -r num; do
    sum=$((sum + num))
    count=$((count + 1))
done < numbers.txt
average=$((sum / count))
echo "Average: $average"
```

49. Test Builtin

Task: Use `[[ ]]` for regex and file tests.

Solution:

```
filename="test.txt"
string="Hello123"
if [[ -r "$filename" && -w "$filename" ]]; then
    echo "File is readable and writable."
fi
if [[ "$string" =~ ^[A-Z][a-z]+[0-9]+$ ]]; then
    echo "String matches the pattern."
fi
```

50. Signal Trapping

Task: Trap Ctrl-C for graceful exit.

Solution:

```
#!/bin/bash
cleanup() {
    echo "Caught interrupt signal. Cleaning up..."
    exit 1
}
trap cleanup INT TERM
echo "Running... Press Ctrl-C to test."
sleep 10
```

51. Here Documents

Task: Generate dynamic file from variables.

Solution:

```
db_host="localhost"
db_user="admin"
cat > config.yml <<EOF
database:
```

```
  host: $db_host
  user: $db_user
  port: 3306
app:
  name: "My App"
EOF
```

52. Process Substitution

Task: Compare output without temp files.

Solution:

```
diff <(ls -1 /dir1) <(ls -1 /dir2)
```

53. Debugging

Task: Demonstrate debugging techniques.

Solution:

```
# Method 1: Simple tracing
set -x
echo "This will be traced"
set +x

# Method 2: Custom DEBUG trap
trap 'echo "DEBUG: executing: $BASH_COMMAND"' DEBUG
echo "Hello"
x=10
echo "World"
trap - DEBUG
```

54. The `exec` Command

Task: Demonstrate replacing shell process.

Solution:

```
#!/bin/bash
echo "This script will replace itself with 'top'"
exec top
echo "This line never executes"
```

55. Read File Line by Line

Task: Most robust method to read file.

Solution:

```
while IFS= read -r line; do
    printf 'Processed: %s\n' "$line"
done < "input_file.txt"
```
