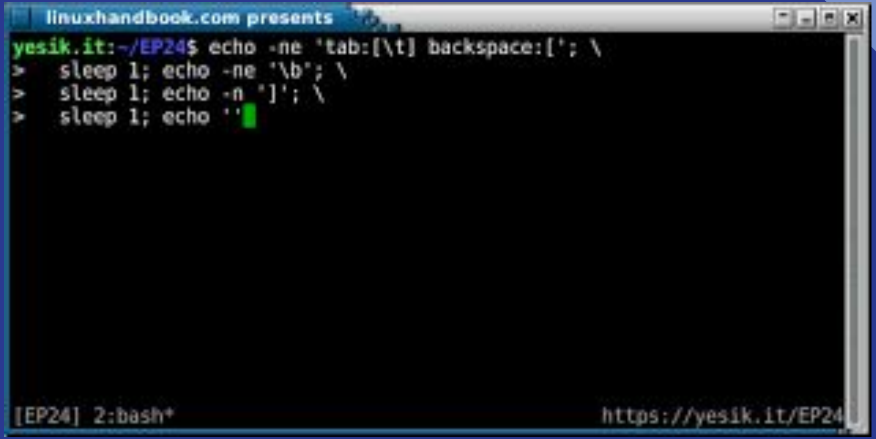
Bash Scripting!
Class 4



```
yesik.it:~/EP24$ echo -ne 'tab:[\t] backspace:['; \
>   sleep 1; echo -ne '\b'; \
>   sleep 1; echo -n ']'; \
>   sleep 1; echo ''
```

```
[EP24] 2:bash*                                    https://yesik.it/EP24
```

Class 4 Overview

- Regular Expressions
- Error handling
- Logging and debugging
- Directory Stack
- FileSystem Operation
- Utility Functions
- Mastering Man Pages
- Practice: Hands-on exercises, code examples

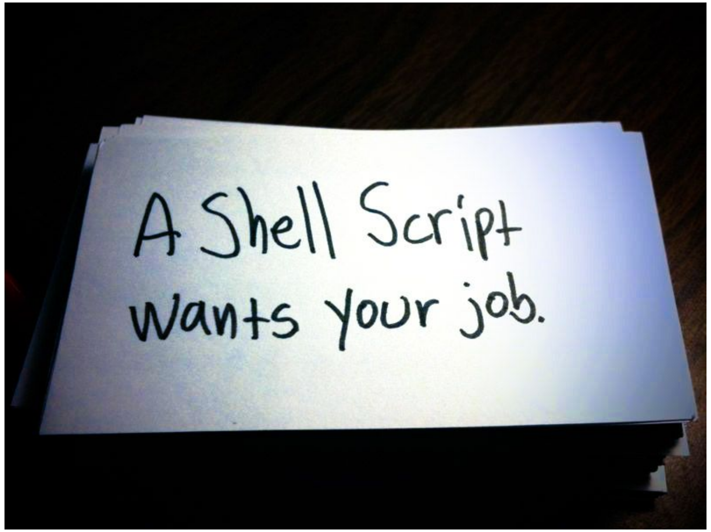


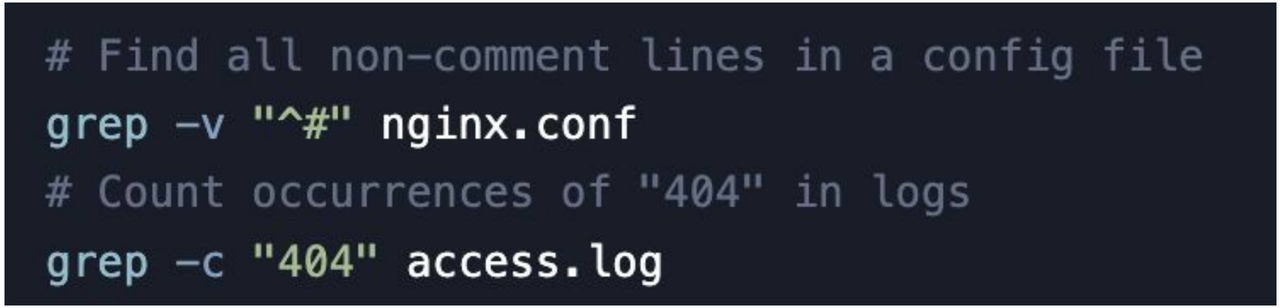
Image credit: https://www.flickr.com/photos/jm3/4814208649

Regular Expressions in Shell

- **Definition: Pattern matching syntax for**
  - text processing, search, filtering
- **DevOps use cases**
  - log analysis
  - data filtering
  - input validation
- **Common patterns: . * ^ $ [] () |**
  - . → any one character
  - ^ → beginning of the line
  - $ → ending of the line
  - * → multiple chars
  - [a-z] → specific range
  - \b → word boundary
- **Tools: grep -E, sed -n, awk**

grep - Global Regular Expression Print

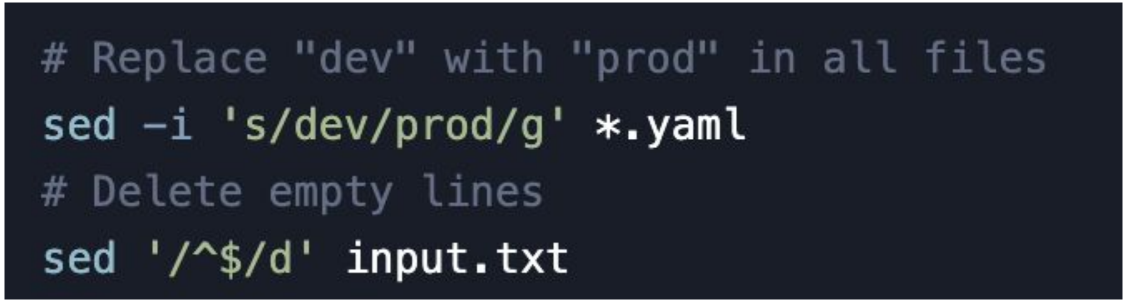
- **Purpose**: Search text/patterns in files or input streams.
- **Key Options**:
  - -i (case-insensitive), -v (invert match), -E (extended regex),
  - -r (recursive), -l (show filenames only), -c (count matches).
- **DevOps Use Cases**:
  - Filter logs (grep "ERROR" app.log),
  - Find config parameters (grep -r "PORT" /etc/).



```
# Find all non-comment lines in a config file
grep -v "^#" nginx.conf
# Count occurrences of "404" in logs
grep -c "404" access.log
```

sed - Stream Editor

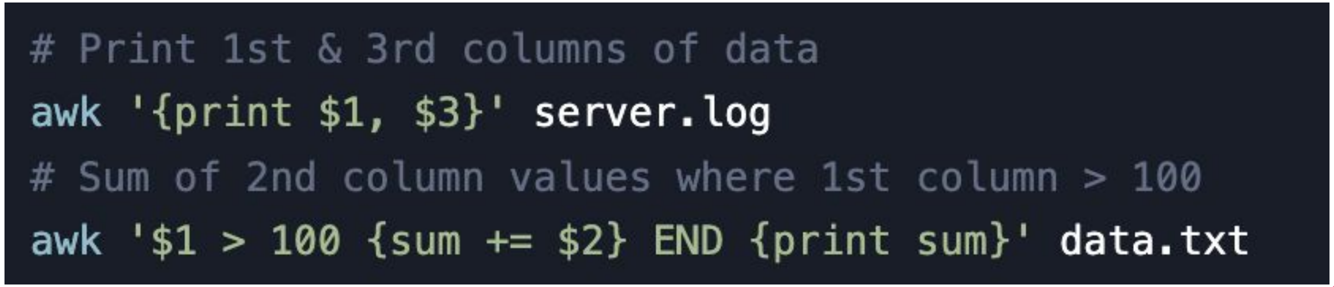
- **Purpose**: Edit text streams (replace, delete, insert) without opening files.
- **Key Features**:
  - s/regex/replacement/flags (substitute),
  - -i (edit file in-place),
  - /d (delete lines), /p (print).
- **DevOps Use Cases**:
  - Bulk replace IPs in configs (sed -i 's/old_ip/new_ip/g' *.conf),
  - Clean log files (remove timestamps).



```
# Replace "dev" with "prod" in all files
sed -i 's/dev/prod/g' *.yaml
# Delete empty lines
sed '/^$/d' input.txt
```

awk - Aho, Weinberger, and Kernighan (initials of its designers)

- **A Test processing powerhouse**
- **Purpose**: Process column-based data & generate reports.
- **Syntax**: awk 'pattern {action}' file
- **Built-in Variables**:
    - NR (row number), NF (fields count), $0 (entire line), $1, $2 (columns).
- **DevOps Use Cases**:
    - Analyze server metrics (CPU, Memory),
    - Format log data (e.g., extract timestamps).



```
# Print 1st & 3rd columns of data
awk '{print $1, $3}' server.log
# Sum of 2nd column values where 1st column > 100
awk '$1 > 100 {sum += $2} END {print sum}' data.txt
```

Comparison & When to Use (grep, sed, awk)

- **grep**: Quick search/filtering (simpler than regex).
- **sed**: Find-and-replace, delete lines (text manipulation).
- **awk**: Column-based processing, calculations, reports.

```
# Top 5 URLs with 404 errors
awk '$9 == 404 {print $7}' access.log | sort | uniq -c | sort -nr | head -5
```

Log Analysis

```
# Replace "Listen 80" with "Listen 8080" in all conf files
sed -i 's/Listen 80/Listen 8080/g' /etc/nginx/*.conf
```

Config Update

```
# Find SSH login attempts in auth.log
grep "Failed password" /var/log/auth.log | awk '{print $11}' | sort | uniq -c
```

Security Check

```
# Find "ERROR" lines, extract 2nd column, count occurrences
grep "ERROR" app.log | awk '{print $2}' | sort | uniq -c
```

Logging

- **What and why?**
    - The process of recording events, activities, or messages generated by a system
    - That may scroll off the screen
    - Who, what, when, where, why something
    - Script may run via cron and others
- **Syslog Standard**
    - Syslog is a standard logging mechanism used in Unix-like operating systems to collect, process, and store log messages generated by various components of the system
    - Centralized Logging
    - Severity Levels - ranging from "emergency" to "debug", elert, warning,
    - Facilities - "auth" for authentication-related messages, "mail" for mail server messages, and "kernel" for messages from the operating system kernel
    - Configurability - Log file location are configurable - var/log/messages, var/log/syslog
- **Generating log messages**
- **Custom logging functions**

Logging - Examples

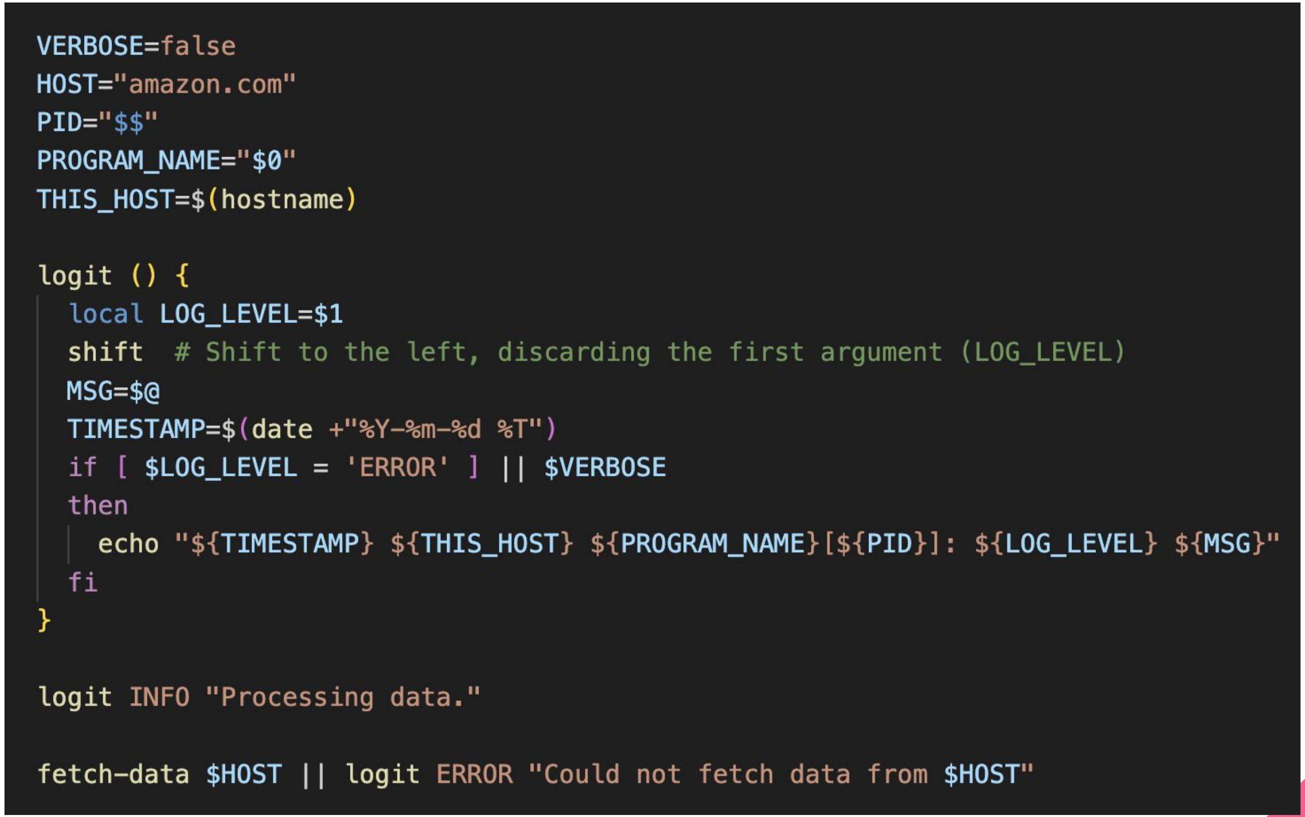
```
#!/bin/bash

logger "Message"
logger -p local0.info "Message"
logger -s -p local0.info "Message"
logger -t myscript -p local0.info "Message"
logger -i -t myscript "Message"
```

Output:

```
Mar  3 19:06:12  root[55028] <Info>: Message
```

- This sends a log message with the content **"Message"** to the syslog daemon. By default, the message is logged with the facility "user" and severity level "notice".
- **-p local0.info** specifies the facility as "local0" and severity level as "info".
- **-s** logs the message to the system console.
- **-t myscript** tags the message with the program name "myscript".
- **-i** Log the message with the process ID included.

Logging - Examples



```
VERBOSE=false
HOST="amazon.com"
PID="$$"
PROGRAM_NAME="$0"
THIS_HOST=$(hostname)

logit () {
  local LOG_LEVEL=$1
  shift  # Shift to the left, discarding the first argument (LOG_LEVEL)
  MSG=$@
  TIMESTAMP=$(date +"%Y-%m-%d %T")
  if [ $LOG_LEVEL = 'ERROR' ] || $VERBOSE
  then
    echo "${TIMESTAMP} ${THIS_HOST} ${PROGRAM_NAME}[${PID}]: ${LOG_LEVEL} ${MSG}"
  fi
}

logit INFO "Processing data."

fetch-data $HOST || logit ERROR "Could not fetch data from $HOST"
```

Output:
2024-03-03 21:31:36 BS847s-MacBook-Pro.local /bin/zsh[45508]: ERROR Could not fetch data from amazon.com

Logging - Examples

- VERBOSE=false: A variable indicating whether verbose logging is enabled or not. By default, it is set to false.
- HOST="google.com": A variable storing the hostname.
- PID="$$": A variable storing the process ID of the script.
- PROGRAM_NAME="$0": A variable storing the name of the script.
- THIS_HOST=$(hostname): A variable storing the hostname of the current machine.
- logit () { ... }: Definition of the logit function, which takes two arguments: LOG_LEVEL and MSG. Inside the function:
    - TIMESTAMP=$(date +"%Y-%m-%d %T"): Variable storing the current timestamp in the format "YYYY-MM-DD HH:MM:SS".
    - The function checks whether the LOG_LEVEL is 'ERROR' or VERBOSE is true. If it is an error message or VERBOSE is true, the log message is printed.
    - The log message includes the timestamp, hostname, script name, process ID, log level, and message.
- logit INFO "Processing data.": Calls the logit function with log level INFO and the message "Processing data.".
- fetch-data $HOST || logit ERROR "Could not fetch data from $HOST": Calls the fetch-data function with the hostname stored in HOST. If the fetch-data function fails (returns a non-zero exit status), it logs an error message.

Debugging

Several reasons why debugging is essential:

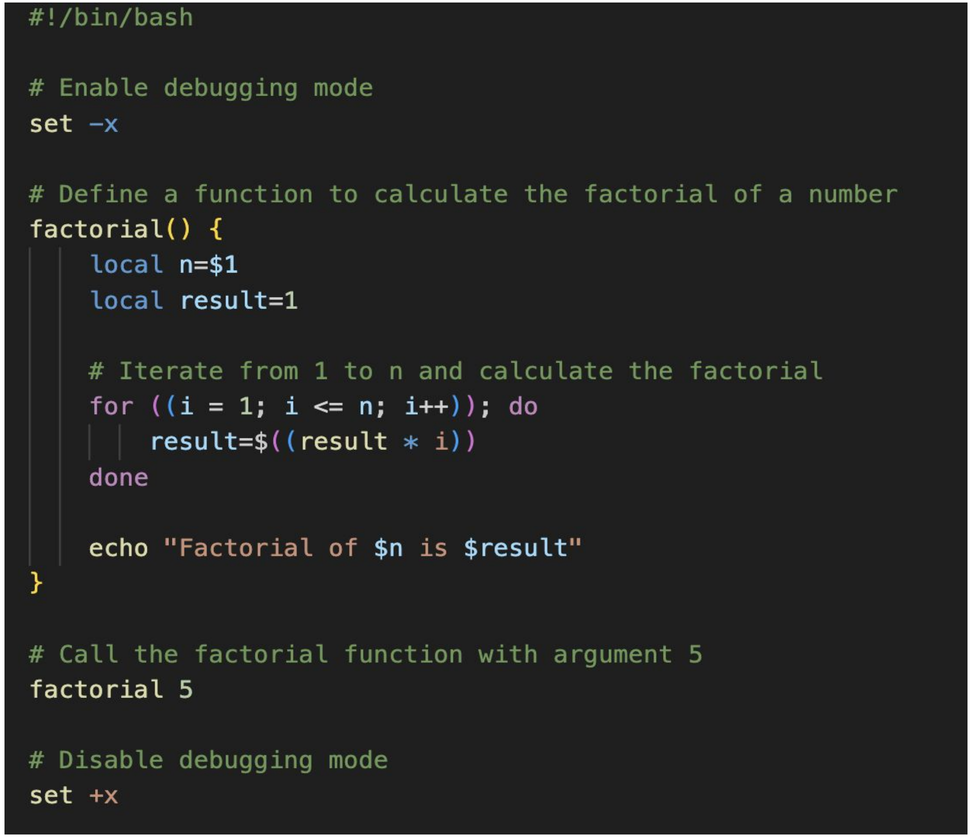
- **Identifying Errors**
- **Examine the inner working of the script**
- **Find out the root cause of unexpected behaviour**
- **Fixing bugs**
- **Testing and Validation**

Debugging

```
#!/bin/bash

# Enable debugging mode
bash -x script.sh

# Or, within the script itself
set -x

# Your script commands here

# Disable debugging mode
set +x
```

- The -x option (or -v) is used to enable debugging mode. When this option is set, Bash displays each command before executing it, allowing you to trace the execution flow and identify any issues in the script.
- Called as **x-trace, tracing,** or **print debugging**

Debugging - Example



```bash
#!/bin/bash

# Enable debugging mode
set -x

# Define a function to calculate the factorial of a number
factorial() {
    local n=$1
    local result=1

    # Iterate from 1 to n and calculate the factorial
    for ((i = 1; i <= n; i++)); do
        result=$((result * i))
    done

    echo "Factorial of $n is $result"
}

# Call the factorial function with argument 5
factorial 5

# Disable debugging mode
set +x
```

- set -x: Enables debugging mode, which causes Bash to display each command before executing
- factorial() function: Defines a function to calculate the factorial of a number.
- Within the function:
  - local variables are used to ensure variable scope.
  - A for loop calculates the factorial of the input number.
  - An echo statement displays the result.
- factorial 5: Calls the factorial function with the argument 5 to calculate the factorial of 5.
- set +x: Disables debugging mode

Debugging - (Built-in)

- **-e**: This option, when set, ensures that the script exits immediately if any command returns a non-zero exit status. It's often used to enforce strict error handling in scripts.

- **-x**: This option, when set, enables debugging mode, where Bash displays each command before executing it. It's useful for tracing the execution flow of the script and identifying any issues.

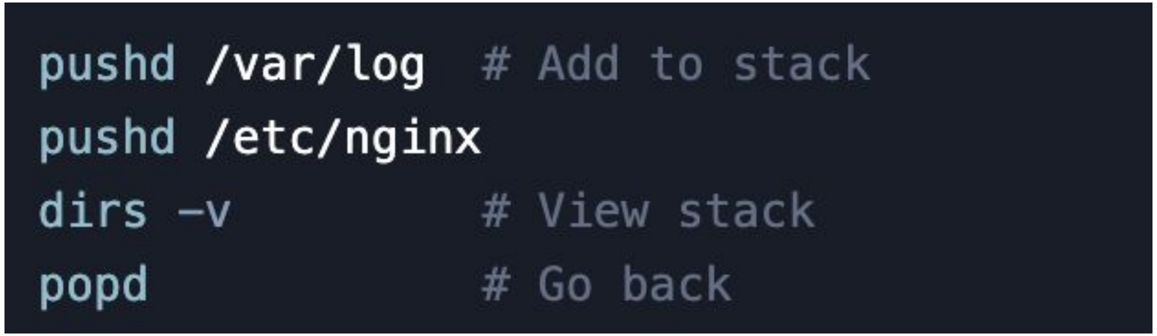
- **-v**: This option, when set, displays shell input lines as they are read, useful for debugging interactive sessions or scripts.

- **-ex or -xe**: These combinations of options enable both -e (exit on error) and -x (debugging mode) simultaneously. It's commonly used for debugging scripts where immediate exit on error and command tracing are desired.

Managing Directory Stack

A directory stack is a LIFO (Last In, First Out) structure used to manage directories efficiently. It allows you to quickly navigate between directories without manually typing long paths every time.

- Commands: **pushd, popd, dirs**
  - pushd → Push a directory onto the stack and switch to it.
  - popd → Pop (remove) the top directory from the stack and switch to the previous one.
  - dirs → Display the current stack of directories.
- Purpose: Navigate multiple directories without typing full paths



```
pushd /var/log   # Add to stack
pushd /etc/nginx
dirs -v          # View stack
popd             # Go back
```

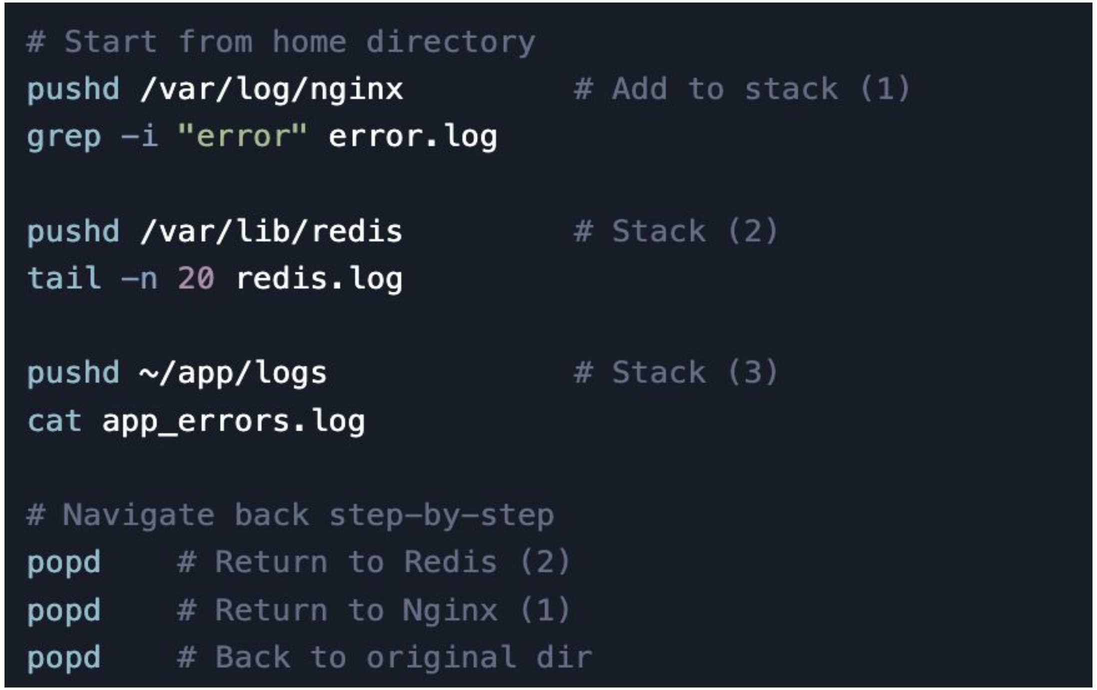
Managing Directory Stack

- **Why DevOps Engineers Care**:
  - Rapid context switching between logs/configs/projects.
  - Simplify scripts requiring multi-directory operations.

- **Real-World Use Cases**:
  - Troubleshooting across multiple service directories.
  - Deploying microservices from different paths.
  - Auditing logs in /var/log/ subdirectories.

Managing Directory Stack (Examples)

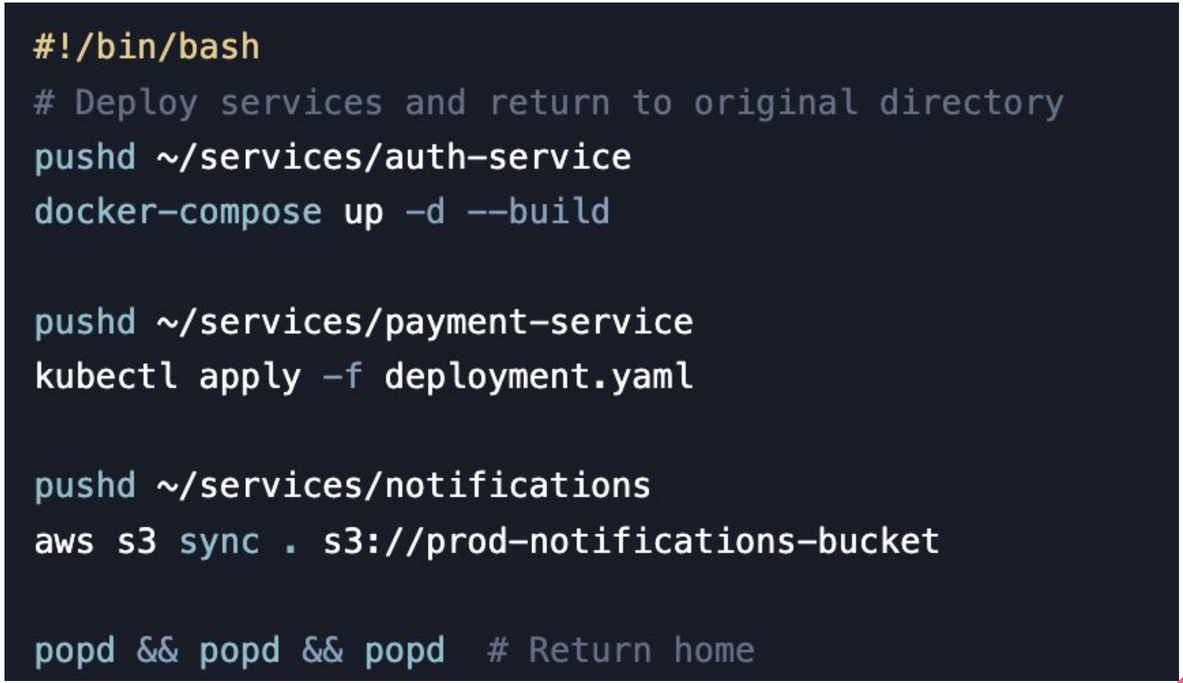
- **Example 1: Multi-Service Log Investigation**
  - **Scenario**: Check error logs across Nginx, Redis, and your app.



```
# Start from home directory
pushd /var/log/nginx         # Add to stack (1)
grep -i "error" error.log

pushd /var/lib/redis         # Stack (2)
tail -n 20 redis.log

pushd ~/app/logs             # Stack (3)
cat app_errors.log

# Navigate back step-by-step
popd    # Return to Redis (2)
popd    # Return to Nginx (1)
popd    # Back to original dir
```

Managing Directory Stack (Examples)

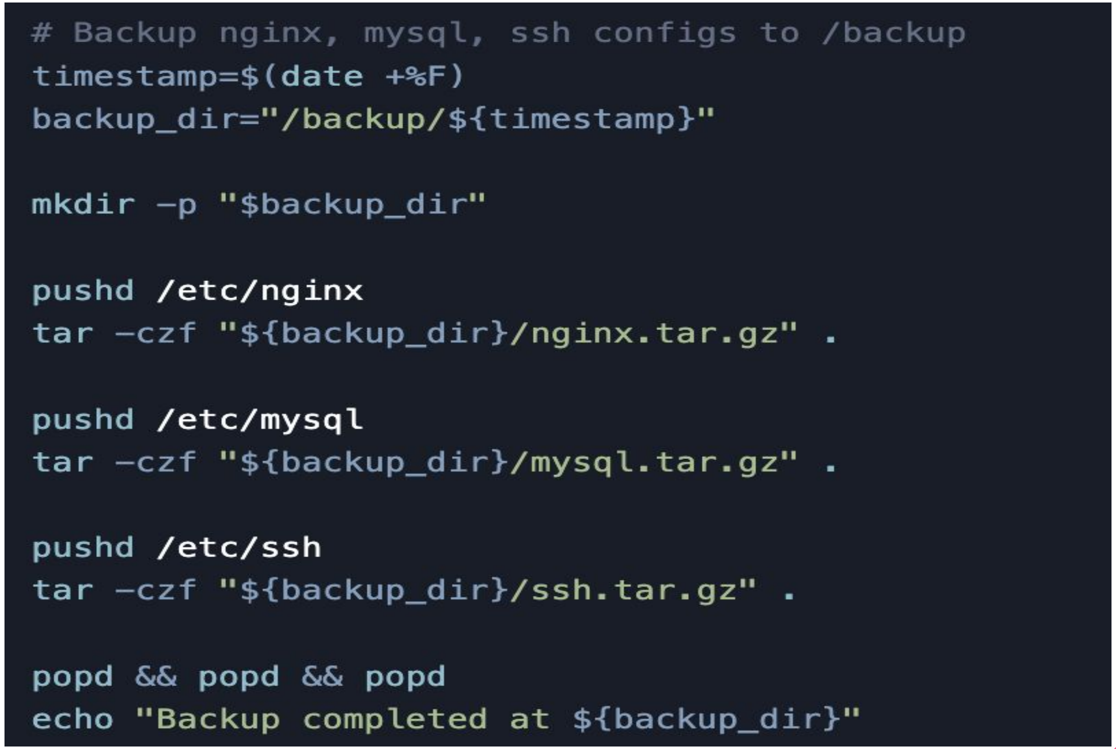
- **Example 2: Scripted Deployment for Microservices**
    - **Scenario**: Deploy 3 microservices in sequence.



```
#!/bin/bash
# Deploy services and return to original directory
pushd ~/services/auth-service
docker-compose up -d --build

pushd ~/services/payment-service
kubectl apply -f deployment.yaml

pushd ~/services/notifications
aws s3 sync . s3://prod-notifications-bucket

popd && popd && popd  # Return home
```

Managing Directory Stack (Examples)

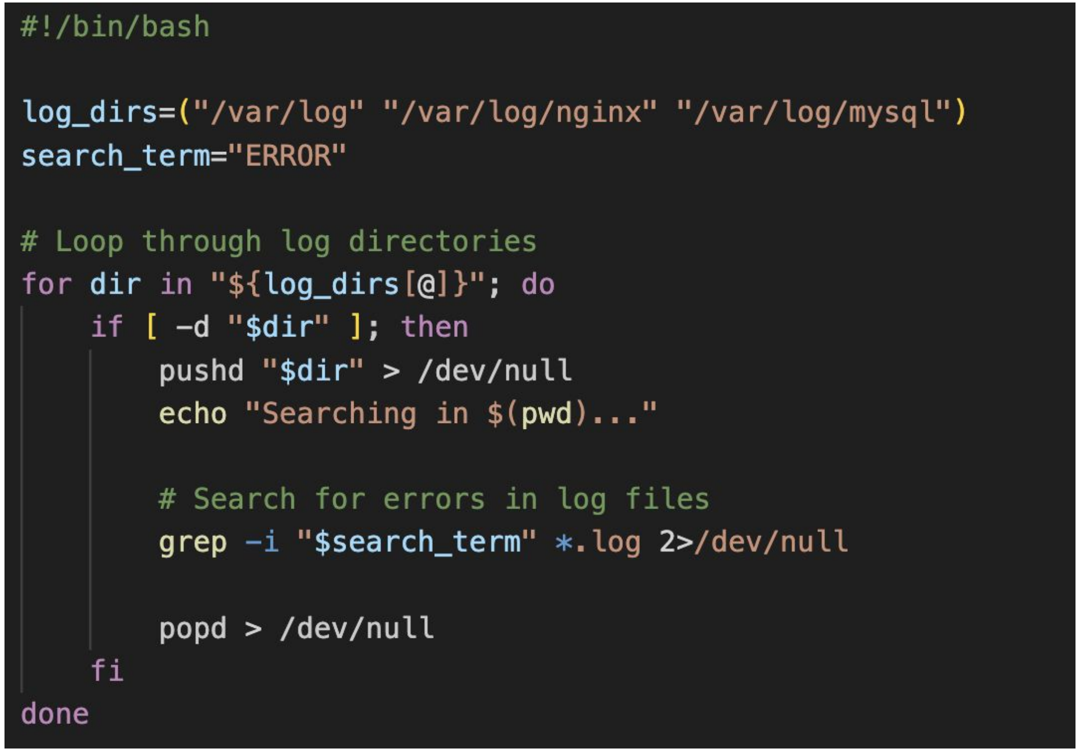
- **Example 3: Automated Backup with Directory Stack**
  - **Scenario**: Backup configs from /etc/ subfolders.



```
# Backup nginx, mysql, ssh configs to /backup
timestamp=$(date +%F)
backup_dir="/backup/${timestamp}"

mkdir -p "$backup_dir"

pushd /etc/nginx
tar -czf "${backup_dir}/nginx.tar.gz" .

pushd /etc/mysql
tar -czf "${backup_dir}/mysql.tar.gz" .

pushd /etc/ssh
tar -czf "${backup_dir}/ssh.tar.gz" .

popd && popd && popd
echo "Backup completed at ${backup_dir}"
```

Managing Directory Stack (Examples)

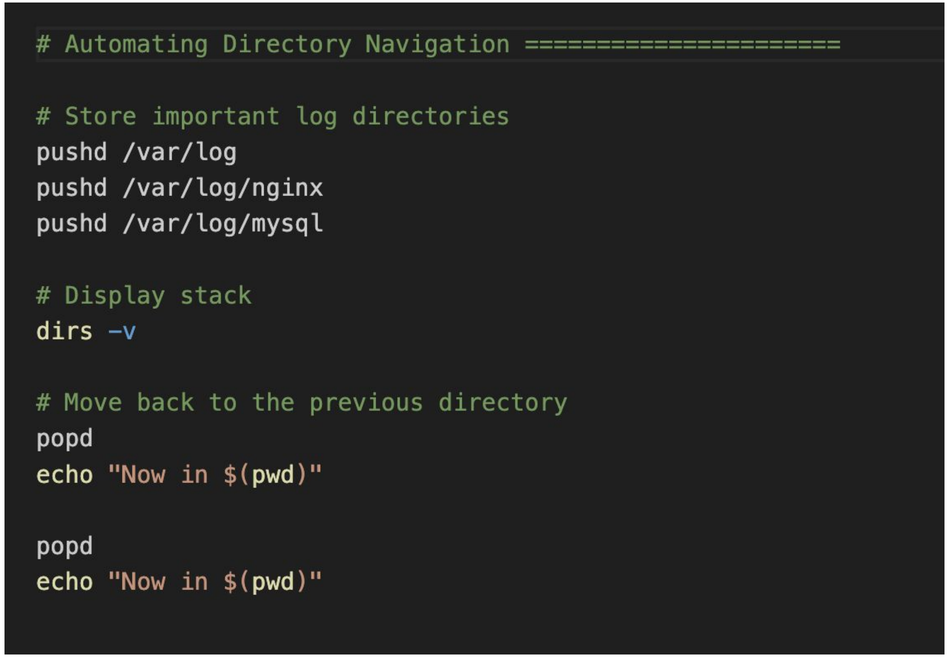
- **Log Analysis Using Directory Stack**
  - **Scenario**: Automatically Switching Between Important Logs



```bash
#!/bin/bash

log_dirs=("/var/log" "/var/log/nginx" "/var/log/mysql")
search_term="ERROR"

# Loop through log directories
for dir in "${log_dirs[@]}"; do
    if [ -d "$dir" ]; then
        pushd "$dir" > /dev/null
        echo "Searching in $(pwd)..."

        # Search for errors in log files
        grep -i "$search_term" *.log 2>/dev/null

        popd > /dev/null
    fi
done
```

Managing Directory Stack (Examples)

- **Navigating Infrastructure Logs**
  - When working with infrastructure logs, DevOps engineers often move between /var/log, /etc/nginx, and application logs. A directory stack makes navigation easier.



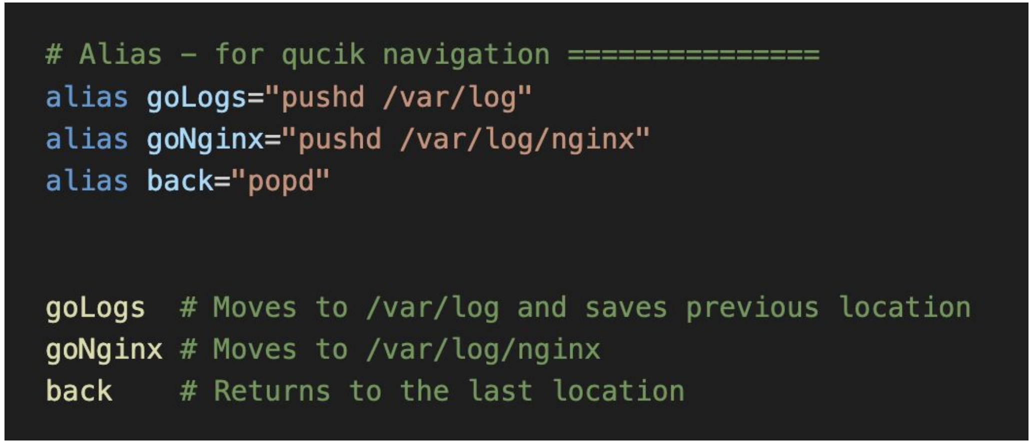
```
# Automating Directory Navigation ======================

# Store important log directories
pushd /var/log
pushd /var/log/nginx
pushd /var/log/mysql

# Display stack
dirs -v

# Move back to the previous directory
popd
echo "Now in $(pwd)"

popd
echo "Now in $(pwd)"
```

Easily switch between nginx, mysql, and /var/log logs without manually navigating each time.

Managing Directory Stack (DevOps Best Practice)

Best Practices for DevOps:

- **Alias for Quick Navigation**:
  - `alias stack='dirs -v'  # Add to ~/.bashrc`
- **Safety Check**: Always use pushd +N to jump to specific stack entries.
- **Scripting Tip**: Use pushd/popd in pairs to avoid stack corruption.



```
# Alias - for qucik navigation ===============
alias goLogs="pushd /var/log"
alias goNginx="pushd /var/log/nginx"
alias back="popd"


goLogs  # Moves to /var/log and saves previous location
goNginx # Moves to /var/log/nginx
back    # Returns to the last location
```

- **Problem**: popd error when stack is empty.
  Fix: Check stack first with dirs -v.
- **Problem**: Relative vs absolute paths.
  Fix: Always use pushd "$(pwd)" in scripts.

Filesystem Operations

- **What**: Create, delete, modify, and organize files/directories programmatically.
- **Why in DevOps**:
  - Automate environment setup (e.g., creating log/config directories).
  - Manage deployments, backups, and cleanups.
  - Handle permissions for security (e.g., Kubernetes secrets, SSH keys).
- **Key Commands**: mkdir, touch, cp, mv, rm, chmod, chown, find, rsync.

ls → List directory contents
cd → Change directory
pwd → Print current directory
mkdir → Create directories
rm → Remove files/directories
cp → Copy files/directories
mv → Move or rename files/directories
touch → create empty files and update the timestamps of existing files
chmod → Change file permissions
chown → Change file ownership
find → Search files and directories
rsync → Synchronize files and directories

Essential Filesystem Commands

```
mkdir -p /app/{logs,config}  # Nested directories
touch /app/logs/app.log      # Create empty file
```

Create

```
cp -r /source /dest             # Recursive copy
mv old.log /archive/            # Move/rename
```

Copy/Move

```
rm -rf /tmp/*                # Caution! Force delete
```

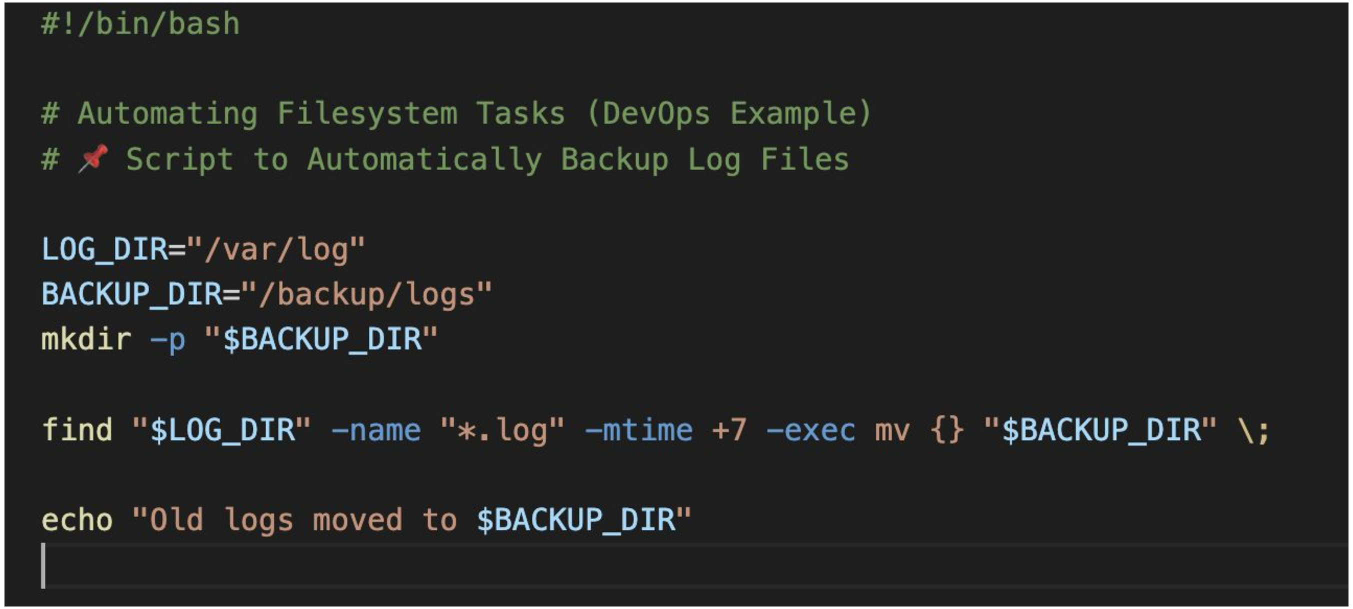
Delete

```
chmod 600 ~/.ssh/id_rsa     # Read/write for owner
chown root:root /app/config # Change owner/group
```

Permissions

Security Tip: Always set proper permissions to avoid unauthorized access.

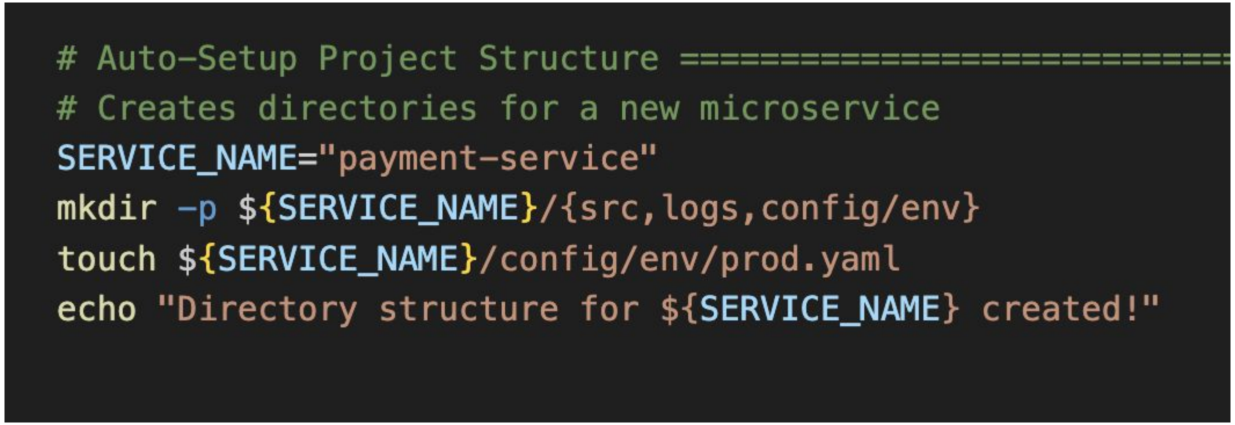
Examples (Automating Filesystem Tasks)



```
#!/bin/bash

# Automating Filesystem Tasks (DevOps Example)
# 📌 Script to Automatically Backup Log Files

LOG_DIR="/var/log"
BACKUP_DIR="/backup/logs"
mkdir -p "$BACKUP_DIR"

find "$LOG_DIR" -name "*.log" -mtime +7 -exec mv {} "$BACKUP_DIR" \;

echo "Old logs moved to $BACKUP_DIR"
```

Purpose: Moves logs older than 7 days to a backup directory for better log management.

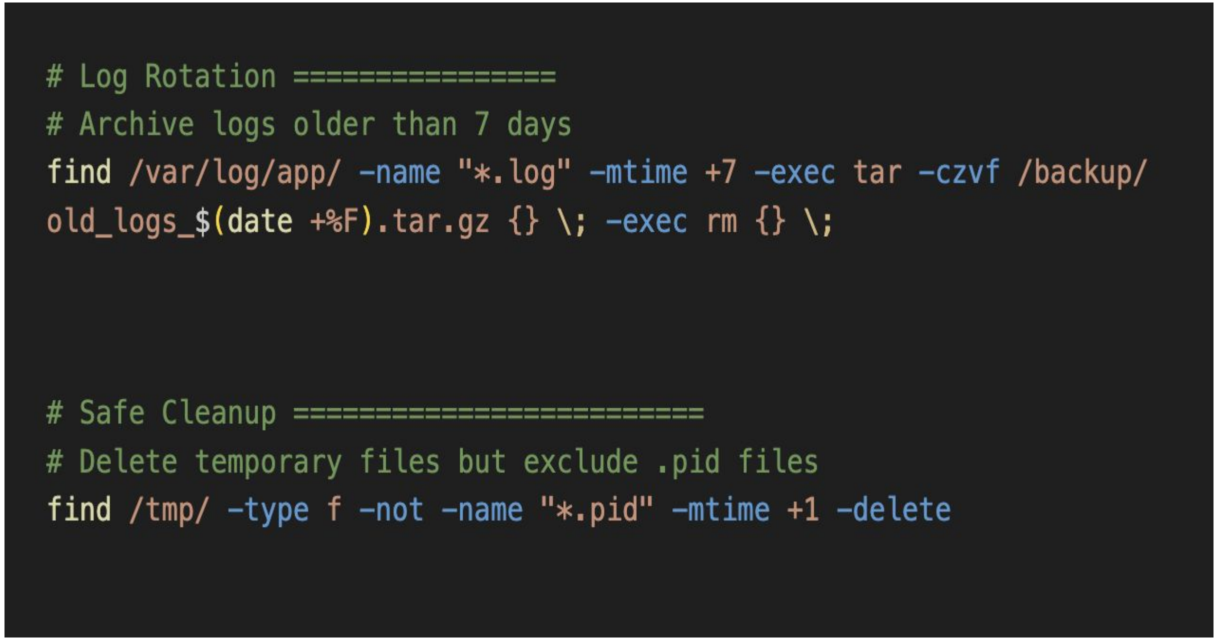
Filesystem Examples (Auto-Setup Project Structure)



```
# Auto-Setup Project Structure ============================
# Creates directories for a new microservice
SERVICE_NAME="payment-service"
mkdir -p ${SERVICE_NAME}/{src,logs,config/env}
touch ${SERVICE_NAME}/config/env/prod.yaml
echo "Directory structure for ${SERVICE_NAME} created!"
```

Purpose: Creates new microservice folder structure named `payment-service`

Filesystem Examples (Log Rotation, Safe Cleanup)



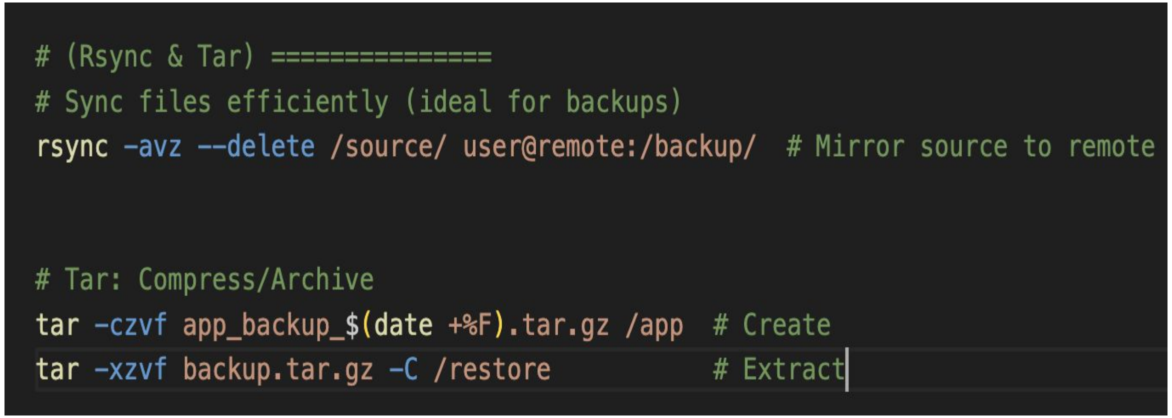
```
# Log Rotation ================
# Archive logs older than 7 days
find /var/log/app/ -name "*.log" -mtime +7 -exec tar -czvf /backup/
old_logs_$(date +%F).tar.gz {} \; -exec rm {} \;


# Safe Cleanup =======================
# Delete temporary files but exclude .pid files
find /tmp/ -type f -not -name "*.pid" -mtime +1 -delete
```

Purpose: Archive logs older than 7 days

Purpose: Delete 1 day old temporary file except .pid file

Advanced Filesystem Operations (Rsync & Tar)

Rsync: Synchronizing files and directories between two locations over a remote shell, or from/to a remote Rsync daemon. Rsync can be used for mirroring data, incremental backups, copying files between systems, and as a replacement for scp , sftp , and cp commands.



```
# (Rsync & Tar) ===============
# Sync files efficiently (ideal for backups)
rsync -avz --delete /source/ user@remote:/backup/  # Mirror source to remote


# Tar: Compress/Archive
tar -czvf app_backup_$(date +%F).tar.gz /app  # Create
tar -xzvf backup.tar.gz -C /restore           # Extract
```

Sync files efficiently (ideal for backups).

Compress the entire /app folder with tar -czvf. Use this archive during deployment.

Syntax:
rsync [options] source [destination]

tar [options] [archive-file] [file or directory to be archived]

Filesystem Operations - Best Practices

- **Do**:
  - Use -i (interactive) with rm/cp in scripts.
  - Check file/dir existence: [-d "/app"] || mkdir /app.
- **Don't**:
  - Use rm -rf / or $(sudo rm -rf /) (💀 infamous Linux joke).
  - Assume paths – always use absolute paths in scripts.

Exercise (Assignment)

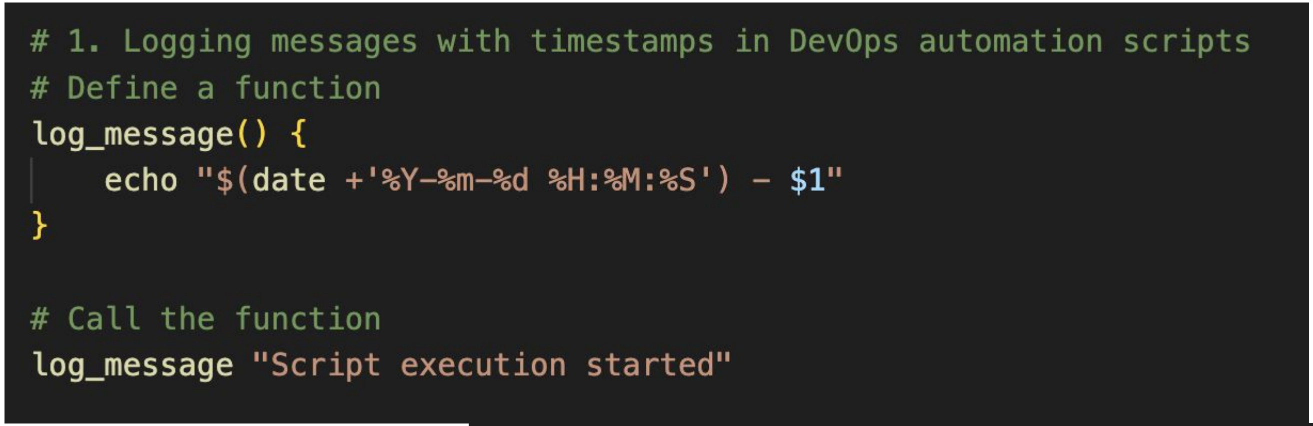
Task: Write a script to:

1. Create **/backup/YEAR-MONTH-DAY** directory.
2. Copy all **.conf** files from **/etc** to this directory.
3. Compress the backup.
4. Delete backups older than 30 days.

Utility Functions

- **Definition**: Reusable code blocks for repetitive tasks (e.g., logging, error handling).
- **Why Use Them**:
  - Reduce code duplication
  - Simplify debugging
  - Organizing codes
  - Improve script readability
  - Making it modular and maintainable
- **DevOps Use Cases**:
  - Logging with timestamps
  - Error notifications (Slack/Email)
  - System monitoring
  - Resource cleanup

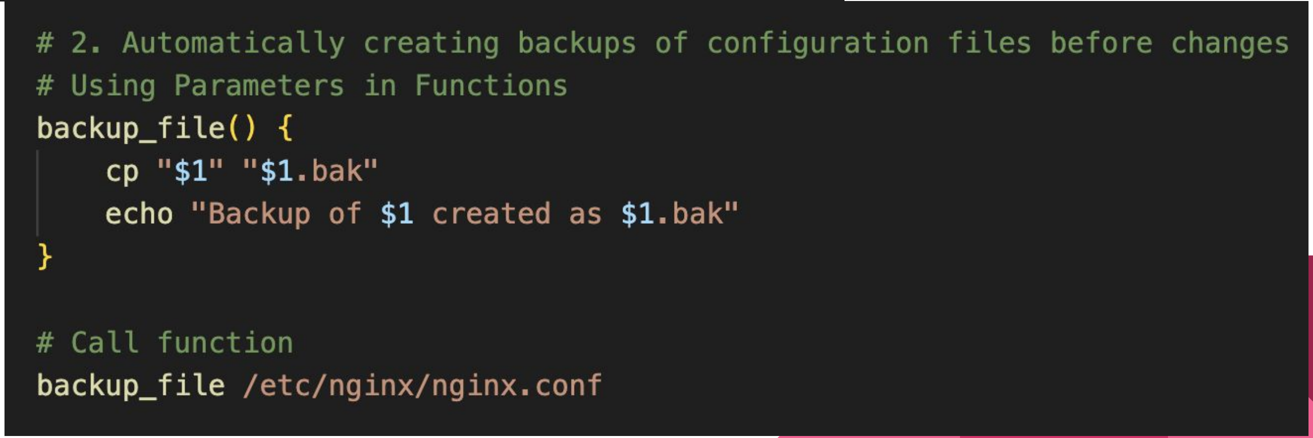
Utility Functions (Examples)



```bash
# 1. Logging messages with timestamps in DevOps automation scripts
# Define a function
log_message() {
    echo "$(date +'%Y-%m-%d %H:%M:%S') - $1"
}

# Call the function
log_message "Script execution started"
```

Output:

```
2025-02-16 17:09:00 -
Script execution started
```

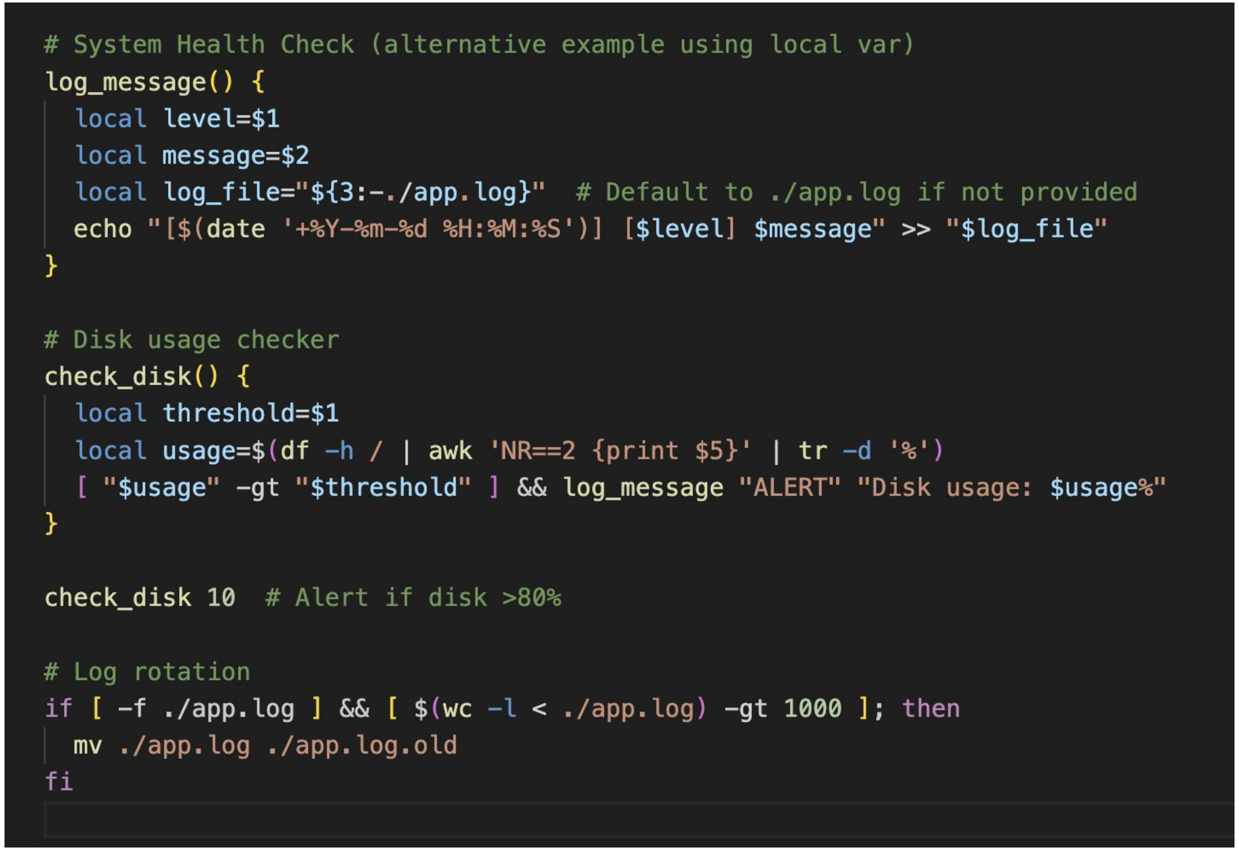


```bash
# 2. Automatically creating backups of configuration files before changes
# Using Parameters in Functions
backup_file() {
    cp "$1" "$1.bak"
    echo "Backup of $1 created as $1.bak"
}

# Call function
backup_file /etc/nginx/nginx.conf
```

Output:

```
Backup of
/etc/nginx/nginx.conf
created as
/etc/nginx/nginx.conf.bak
```

Utility Functions (Examples)



```
# System Health Check (alternative example using local var)
log_message() {
  local level=$1
  local message=$2
  local log_file="${3:-./app.log}"  # Default to ./app.log if not provided
  echo "[$(date '+%Y-%m-%d %H:%M:%S')] [$level] $message" >> "$log_file"
}

# Disk usage checker
check_disk() {
  local threshold=$1
  local usage=$(df -h / | awk 'NR==2 {print $5}' | tr -d '%')
  [ "$usage" -gt "$threshold" ] && log_message "ALERT" "Disk usage: $usage%"
}

check_disk 10  # Alert if disk >80%

# Log rotation
if [ -f ./app.log ] && [ $(wc -l < ./app.log) -gt 1000 ]; then
  mv ./app.log ./app.log.old
fi
```

If you want the log file path to be configurable, you can pass it as an argument
log_message "ALERT" "Disk usage: $usage%" "/var/log/myapp.log"

$usage stores the percentage rate of disk usage

For long-running scripts, you might want to rotate logs to avoid large files

Man Page

- **What**: Manual pages (man pages) are the built-in documentation for Linux commands and tools.
- **Why Master Them**:
  - Quickly learn command syntax, options, and examples.
  - Troubleshoot errors and understand system behavior.
  - Essential for DevOps engineers working with Linux systems.
- **Accessing Man Pages**:

```
man <command>  # Example: man ls
```

Man Page (Navigation and Sections)

- **Navigation Keys:**
  - Spacebar or Page Down: Scroll down.
  - b or Page Up: Scroll up.
  - /keyword: Search for a keyword (e.g., /option).
  - n/N: Jump to next/previous match.
  - q: Quit the man page.

- **Sections:**
  - 1: User commands (e.g., man 1 ls).
  - 5: File formats (e.g., man 5 passwd).
  - 8: System administration commands (e.g., man 8 ifconfig).

Man Page (Searching)

- **Search All Man Pages:**

```
man -k <keyword>  # Example: man -k "network"
```

- **Search Within a Man Page:**
  - Use / followed by the keyword (e.g., /option)
  - Press n to go to the next match

- **Examples:**

```
man -k "copy"      # Find all commands related to "copy"
man -k "process"   # Find commands related to "process"
```

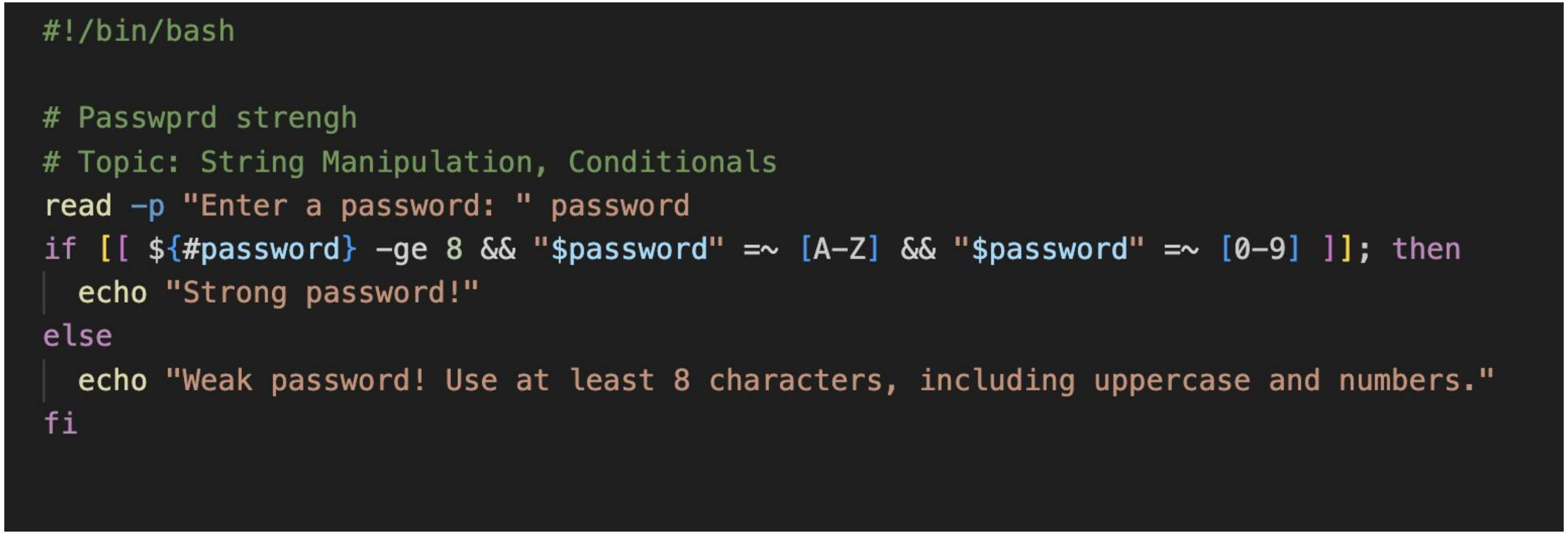
Man Page (Sections)

1 → User Commands (ls, cd, echo)
2 → System Calls (open, read, write)
3 → Library Functions (printf, malloc)
4 → Special Files (/dev/null, /proc)
5 → File Formats & Configurations (/etc/passwd, crontab)
6 → Games (fortune, sl)
7 → Miscellaneous (regex, ascii)
8 → System Admin Commands (iptables, systemctl)

More details: https://linux.die.net/man/

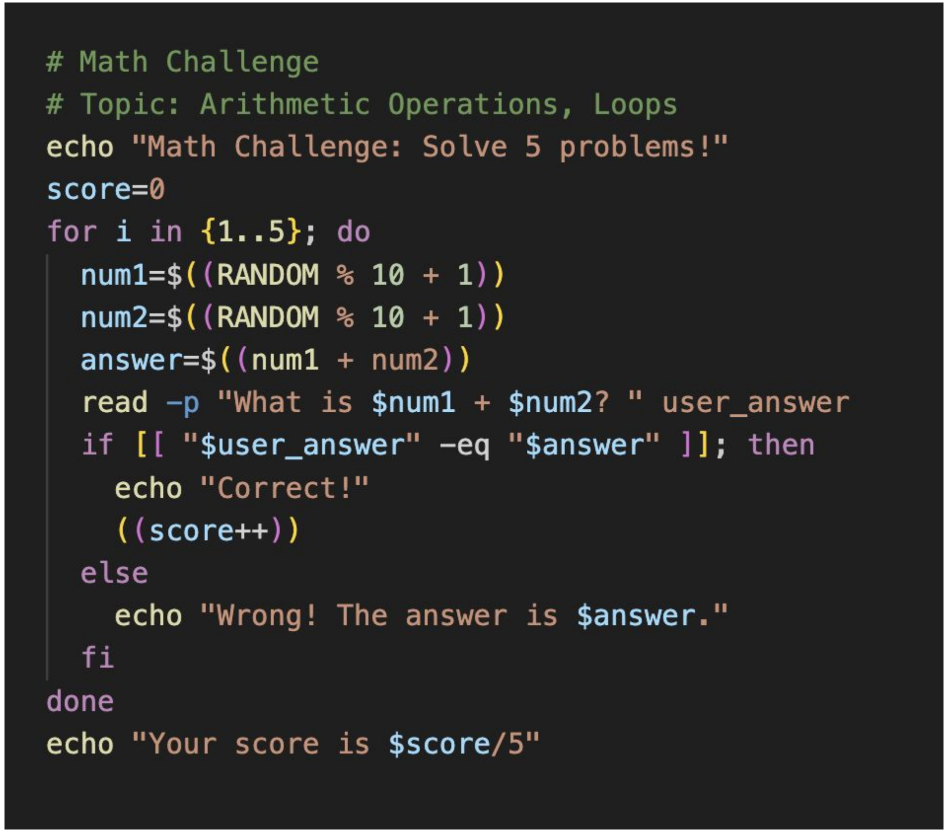
- NAME: Command name and brief description.
- SYNOPSIS: Command syntax.
- DESCRIPTION: Detailed explanation.
- OPTIONS: List of command options.
- EXAMPLES: Usage examples.
- SEE ALSO: Related commands or documentation.

Games (String Manipulation, Conditionals)



```
#!/bin/bash

# Passwprd strengh
# Topic: String Manipulation, Conditionals
read -p "Enter a password: " password
if [[ ${#password} -ge 8 && "$password" =~ [A-Z] && "$password" =~ [0-9] ]]; then
  echo "Strong password!"
else
  echo "Weak password! Use at least 8 characters, including uppercase and numbers."
fi
```

Games (Arithmetic Operations, Loops)



```
# Math Challenge
# Topic: Arithmetic Operations, Loops
echo "Math Challenge: Solve 5 problems!"
score=0
for i in {1..5}; do
  num1=$((RANDOM % 10 + 1))
  num2=$((RANDOM % 10 + 1))
  answer=$((num1 + num2))
  read -p "What is $num1 + $num2? " user_answer
  if [[ "$user_answer" -eq "$answer" ]]; then
    echo "Correct!"
    ((score++))
  else
    echo "Wrong! The answer is $answer."
  fi
done
echo "Your score is $score/5"
```

Live Quiz Competition

Thank you!