Table of Contents

1. Linux Command Cheat Sheet

1.1 Navigation & Exploration

`cd /path` - Change directory
`ls -l /dir` - List files with details
`ls -la /dir` - List all files (incl. hidden)
`pwd` - Print current directory

1.2 File & Directory Management

`cp src dest` - Copy files/dirs
`mkdir name` - Create directory
`mv old new` - Move/rename files
`rm file` - Remove files/dirs
`rm -f file` - Force remove
`rmdir name` - Remove empty dir
`touch file` - Create/update file

1.3 File Viewing & Editing

`cat file` - Display contents
`head -n 5 file` - Show start
`less file` - View page by page
`more file` - Simpler paging
`nano file` - Edit in nano
`tail -n 5 file` - Show end
`tail -f file` - Follow updates
- `vi file` - Edit in vi
- `vim file` - Edit in vim

### 1.4 Permissions & Ownership

`chmod 755 file` - Change permissions
`chmod u+x file` - Add user execute
`chown user file` - Change owner
`ls -l file` - Show permissions

1.5 System Information

`df -h` - Disk space
`free -h` - Memory usage
`ps aux` - List processes
`top` - Monitor processes
`uname -a` - System info
`uptime` - Runtime
`whoami` - Current user

1.6 File Content Manipulation

`grep "pattern" file` - Search text
`sort file` - Sort lines
`uniq file` - Remove duplicates
`wc -l file` - Count lines

1.7 Process Management

`bg` - Run in background
`fg` - Bring to foreground
`jobs` - List background jobs
`kill 1234` - Terminate by ID
`killall name` - Terminate by name

1.8 Networking

`curl http://url` - Fetch data
`ifconfig` - Network config (older)
`ip addr` - Network settings
`netstat -tulpn` - Connections
`ping host` - Test reachability
`wget http://url` - Download

1.9 Package Management

`apt install pkg` - Debian/Ubuntu
`dnf install pkg` - Fedora
`pacman -S pkg` - Arch
`yum install pkg` - CentOS (older)

1.10 Searching

`find /path -name "file"` - Search files

1.11 Miscellaneous

`clear` - Clear screen

`echo "text"` - Print text

`history` - Show command history

`man cmd` - Show manual

`sudo cmd` - Run as superuser

2. Docker Cheat Sheet

Docker build –file [custom docker file name] –t <image-name>:tag . (akhri dot context ka hai kay files kahan say milni hai)

docker run -p 127.0.0.1:8000:8000 test:latest (-p actually port define karta hai kay kis port par chalani hai and –d agr laga daeyn to
wo as a background service par chalay gi without attached terminal)

docker build –target build –t hello . ham yeh wali command use kar sktay hain multistaged main kisi specific point par build ko stop karnay kay liyeh.

docker build –f –Dockerfile –no-cache –target stage2 .

docker login –u username –p password (only to login for docker hub)

To map Docker image port at some port run → sudo docker run -p port:port myfirst:v0.0.1

To push docker image to docker hub we first tag image → sudo docker tag first-app:v0.0.1 syhaiderali/first-app:v0.0.1
sudo docker push syhaiderali/first-app:v0.0.1

2.1 Container Management

`docker --version` - Check version

`docker info` - System info

`docker run -d img` - Start container

`docker ps -a` - List containers

`docker images` - List images

`docker exec -it ctr bash` - Run command

`docker stop ctr` - Stop container

`docker rm ctr` - Remove container

`docker rmi img` - Remove image

2.2 Dockerfile Instructions

`FROM img` - Base image

`RUN cmd` - Execute command

`COPY src dest` - Copy files

`CMD ["cmd"]` - Default command

`EXPOSE port` - Declare port

`VOLUME /path` - Create volume

2.3 Docker Compose

`docker compose up -d` - Start services

`docker compose down` - Stop services

`docker compose build` - Build images

`docker compose ps` - List containers

`docker compose logs` - View logs

2.4 Docker Hub

`docker login` - Log in

`docker tag img user/repo:tag` - Tag image

`docker push user/repo:tag` - Upload

`docker pull user/repo:tag` - Download

2.5 Build

`docker build -t img:tag .` - Build image

`docker build --no-cache -t img .` - Build without cache

3. Project Setup: `projectweb`

3.1 Project Overview

Owner: zaingondal717

Name: projectweb

Services:

Frontend: React (Dockerized)

Backend: Node.js + Express + MongoDB

Database: MongoDB

Image Tags:

`ghcr.io/zaingondal717/projectweb_<service>:${{ github.sha }}` (or `latest` for simplicity)

3.3 Frontend (React)

Setup:

`mkdir -p projectweb/frontend && cd frontend`

`npx create-react-app . && npm install`

Dockerfile:

```
dockerfile
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
RUN npm install -g serve
EXPOSE 3000
CMD ["serve", "-s", "build", "-l", "3000"]
```

Build & Run:

- `docker build -t

ghcr.io/zaingondal717/projectweb_frontend:latest .`

  - `docker run -d -p 3001:3000

ghcr.io/zaingondal717/projectweb_frontend:latest`

3.4 Backend (Node.js)

- **Setup:**

  - `mkdir -p projectweb/backend && cd backend`

  - `npm init -y && npm install express mongoose`

- ****Dockerfile**:**

```
  FROM node:18
  WORKDIR /app
  COPY package*.json ./
  RUN npm install
  COPY . .
  EXPOSE 3000
  CMD ["node", "index.js"]
```

  ****index.js:****

```
  const express = require('express');
  const mongoose = require('mongoose');
  const app = express();
  const mongoUri = process.env.MONGO_URI ||
'mongodb://localhost:27017/mydb';
  mongoose.connect(mongoUri, { useNewUrlParser: true,
useUnifiedTopology: true });
  app.get('/', (req, res) => res.send('Hello from backend'));
  app.listen(3000, () => console.log('Backend on 3000'));
```

- **Build & Run:**

  - `docker build -t

ghcr.io/zaingondal717/projectweb_backend:latest .`

  - `docker run -d -p 3000:3000

ghcr.io/zaingondal717/projectweb_backend:latest`

3.5 GitHub Actions CI/CD

```
name: CI/CD Pipeline
on:
  push:
    branches: [master]
  pull_request:
    branches: [master]
jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:13
        env:
          POSTGRES_USER: postgres
          POSTGRES_PASSWORD: sain
          POSTGRES_DB: devop
        ports:
          - 5432:5432
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
    env:
      DATABASE_URL:
postgres://postgres:sain@localhost:5432/devop
      SECRET_KEY: ${{ secrets.SECRET_KEY }}
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'
      - name: Create log directory
        run: mkdir -p Backend/log
      - name: Install PostgreSQL dependencies
        run: |
          sudo apt-get update
          sudo apt-get install -y libpq-dev postgresql-client
      - name: Install backend dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r Backend/requirements.txt
      - name: Wait for PostgreSQL to be ready
        run: |
          until pg_isready -h localhost -p 5432; do
            echo "Waiting for PostgreSQL to be ready..."
            sleep 1
          done
          echo "PostgreSQL is ready."
      - name: Run Django migrations
        run: |
          cd Backend
          python manage.py makemigrations
          python manage.py migrate
      - name: Run backend tests
        run: |
          cd Backend
          python manage.py test
      - name: Install frontend dependencies
        run: |
          cd frontend
          npm install
      # - name: Run frontend tests
      #   run: |
      #     cd frontend
      #     npm test -- --watchAll=false
  build-and-push-docker:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2
    - name: Log in to Docker Hub
      uses: docker/login-action@v3
      with:
        username: ${{ secrets.DOCKERHUB_USERNAME }}
        password: ${{ secrets.DOCKERHUB_TOKEN }}
    - name: Build and push Docker image
      uses: docker/build-push-action@v5
      with:
        context: ./Backend
        file: ./Backend/Dockerfile
        push: true
        tags: |
          sainsuresh/dev-op-practice-project:latest
          sainsuresh/dev-op-practice-project:${{ github.sha }}
```

3.6 Docker Compose

- **File:** `docker-compose.yml`

```
  version: '3'
  services:
    backend:
      image: ghcr.io/zaingondal717/projectweb_backend:lates
      ports: ["3000:3000"]
      environment: { MONGO_URI:
"mongodb://mongo:27017/mydb" }
    frontend:
      image: ghcr.io/zaingondal717/projectweb_frontend:lates
      ports: ["3001:3000"]
    mongo:
      image: mongo:latest
      ports: ["27017:27017"]
```

Suresh's Docker-Compose.yml:

```
version: "3.8"
services:
  db:
    image: postgres:13
    container_name: myapp_db
    environment:
      POSTGRES_DB: devop
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: sain
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - myapp_network
  backend:
    build:
      context: ./Backend
    container_name: myapp_backend
    environment:
      - DATABASE_URL=postgres://postgres:sain@db:5432/devop
    ports:
      - "8000:8000"
    depends_on:
      - db
    networks:
      - myapp_network
  frontend:
    build:
      context: ./frontend
    container_name: myapp_frontend
    ports:
    - "3000:3000"
    networks:
      - myapp_network
networks:
  myapp_network:
    driver: bridge
volumes:
  postgres_data:
    driver: local
```

- **Run:** `docker compose up -d`

3.7 Additional Notes

- **Local Dev Without Docker:**
  - MongoDB: `docker run -d -p 27017:27017 mongo:latest`
  - Frontend: `cd frontend && PORT=3001 npm start`
  - Backend: `cd backend && node index.js`
- **Local Dev With Docker:** Use `docker-compose.dev.yml` (builds locally).
- **Deployment:** Update `docker-compose.yml` with SHAs, then `docker compose up -d`.

3.8 Questions:

Q1: You want to create a Dockerfile for a Python web app. It should:

- Use Python 3.9
- Install dependencies from `requirements.txt`
- Set working directory as `/app`
- Expose port 5000

Run app.py when the container starts

Answer:

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "app.py"]
```

Q2: How can you ensure that the container runs only after the database service is ready?

Answer:Use HEALTHCHECK to wait for the DB:

```
HEALTHCHECK CMD curl --fail http://db:5432 || exit 1
```

Q4: Why use ENTRYPOINT instead of CMD?

Answer:

– ENTRYPOINT ensures that the main process cannot be overridden when running docker run.

– CMD can be easily overridden with command-line arguments.

3.9 Bash, Shell Scripting:

Shebang & Script Execution

Every Bash script **must** start with a **shebang (`#!`)** to define the interpreter:

```
#!/bin/bash  # This script runs with Bash shell
```

How to Run a Script

```
chmod +x script.sh  # Give execute permissions
./script.sh  # Run the script
```

Variables in Bash

Define Variables

```
name="Alice"
age=25
```

Use Variables

```
echo "Hello, my name is $name and I am $age
years old."
```

Read User Input

```
read -p "Enter your name: " user_name
echo "Welcome, $user_name!"
```

Command Substitution (`$(command)`)

```
current_date=$(date)
echo "Today's date is $current_date"
```

Environment Variables

```
echo "Home directory: $HOME"
echo "Current user: $USER"
```

Conditional Statements

`if` Statements

```
if [ $age -gt 18 ]; then
    echo "You are an adult."
fi
```

`if-else` Statement

```
if [ $age -ge 18 ]; then
    echo "You can vote."
else
    echo "You are too young to vote."
fi
```

`elif` (Else If) Statement

```
if [ $age -lt 13 ]; then
    echo "You are a child."
elif [ $age -lt 20 ]; then
    echo "You are a teenager."
else
    echo "You are an adult."
fi
```

Comparing Strings

```
if [ "$name" == "Alice" ]; then
    echo "Hello Alice!"
fi
```

Loops

For Loop

```
for i in 1 2 3 4 5; do
    echo "Number: $i"
done
```

While Loop

```
count=1
while [ $count -le 5 ]; do
    echo "Iteration $count"
    ((count++))
done
```

Until Loop (Runs Until Condition is True)

```
count=1
until [ $count -gt 5 ]; do
    echo "Iteration $count"
    ((count++))
done
```

Loop Over Files

```
for file in *.txt; do
    echo "Processing $file..."
done
```

Functions in Bash

Basic Function

```
greet() {
    echo "Hello, $1!"
}
greet "Alice"
```

 `$1` refers to the first argument passed to the function.

Function with Multiple Arguments

```
sum() {
    echo "Sum: $(($1 + $2))"
}
sum 5 10
```

Returning Values

```
multiply() {
    echo $(($1 * $2))
}
result=$(multiply 4 5)
echo "Multiplication result: $result"
```

File Handling

Read a File Line by Line

```
while IFS= read -r line; do
    echo "$line"
done < file.txt
```

Write to a File

```
echo "Hello World" > output.txt  # Overwrites
file
echo "Appended line" >> output.txt  # Appends
to file
```

Check If a File Exists

```
if [ -f "file.txt" ]; then
    echo "File exists."
fi
```

Process Management

Run a Command in Background

```
./long_process.sh &
```

Kill a Process

```
kill $(pidof process_name)
```

Check Running Processes

```
ps aux | grep process_name
```

Debugging Bash Scripts

Run Script in Debug Mode

```
bash -x script.sh
```

Enable Debugging Inside Script

```
set -x  # Start debugging
echo "Debugging mode enabled"
set +x  # Stop debugging
```

Useful One-Liners

Check if a Package is Installed

```
dpkg -l | grep package_name
```

Find and Delete Files Larger Than 100MB

```
find /path/to/dir -type f -size +100M -exec rm
-rf {} \;
```

Monitor Log File in Real-Time

```
tail -f /var/log/syslog
```

Find and Replace Text in a File

```
sed -i 's/old-text/new-text/g' file.txt
```

Extract Column from CSV

```
cut -d ',' -f2 data.csv
```

Scenario-Based Questions

Q1: How do you create a script that automatically backs up a directory every hour?

 Answer:

```
#!/bin/bash
src="/home/user/documents"
dest="/backup/documents_$(date +%F_%T).tar.gz"
tar -czf "$dest" "$src"
echo "Backup completed: $dest"
```

➡ Then, schedule it with `cron`:

```
crontab -e
0 * * * * /path/to/backup_script.sh
```

Q2: How do you check if a website is online using Bash?

 Answer:

```
#!/bin/bash
URL="https://example.com"
if curl -s --head --request GET $URL | grep
"200 OK" > /dev/null; then
    echo "Website is online"
else
    echo "Website is down"
fi
```

Q3: How do you create a script that renames all `.txt` files in a directory by adding `_backup` to their name?

 Answer:

```
#!/bin/bash
for file in *.txt; do
    mv "$file" "${file%.txt}_backup.txt"
done
```

Q4: How do you write a script that checks system memory usage and alerts if usage exceeds 90%?

 Answer:

```
#!/bin/bash
mem_usage=$(free | awk '/Mem/{printf "%.2f",
$3/$2 * 100}')
if (( $(echo "$mem_usage > 90" | bc -l) ));
then
    echo "ALERT: Memory usage at
${mem_usage}%!"
fi
```

Q5: How do you automate user creation in Linux using a Bash script?

 Answer:

```
#!/bin/bash
read -p "Enter new username: " new_user
sudo useradd -m $new_user
echo "User $new_user created successfully."
```

Q6: How do you write a script that finds and deletes all log files older than 7 days?

 Answer:

```
#!/bin/bash
find /var/log -name "*.log" -type f -mtime +7 -
exec rm -f {} \;
echo "Deleted all log files older than 7 days."
```

Q7: How do you create a script that monitors CPU usage and sends an alert if usage exceeds 80%?

 Answer:

```
#!/bin/bash
cpu_usage=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')
threshold=80.0

if (( $(echo "$cpu_usage > $threshold" | bc -l) )); then
    echo "ALERT: CPU usage is at ${cpu_usage}%!" | mail -s
"High CPU Usage Alert" user@example.com
fi
```

4.0 Docker Compose Config:

An Basic Example:

```
version: '3.9'
services:
  hello-world:
    image: hello-world:latest
```

Spec: Build:

```
services:
 web:
  # Build from Dockerfile
  build: .
  # Build arguments.
  args:
   APP_HOME: app
  # Build from custom Dockerfile
  build:
   context: ./dir
   dockerfile: Dockerfile.dev
  # Build image.
  image: debian
  image: ubuntu
  image: ubuntu:20.04
```

Network

```
services:
 web:
  # Set container network mode.
  network_mode: "host"
  network_mode: "none"
  network_mode: "service:[service name]"
  # Define the networks that service containers are attached
to.
  networks:
   - some-network
   - other-network

  # Expose container ports.
  ports:
   - "3000"
   - "3000-3005"
   - "8000:8000"
   - "9090-9091:8080-8081"
   - "49100:22"
   - "127.0.0.1:8001:8001"
   - "127.0.0.1:5000-5010:5000-5010"
   - "6060:6060/udp"
  # Define dns server.
  dns: 8.8.8.8
  # Define custom DNS search domains to set on container
network interface configuration.
  dns_search: example.com
  # List custom DNS options to be passed to the container's
DNS resolver.
  dns_opt:
   - use-vc
   - no tld-query
  # Defines a network link to containers in another service.
  links:
   - db
   - db:database
   - redis
```

Environment Variable

```
services:
 web:
  # Define environment variables.
  environment:
   RACK_ENV: development
   SHOW: "true"
   USER_INPUT:
   COMPOSE_PROJECT_NAME: "foo"

  # Define environment variables from file.
  env_file: .env
  env_file:
   - ./a.env
   - ./b.env
```

Commands in docker-compose:

```
services:
 web:
  # Start up command, which overrides the image default
command.
  command: echo "I'm running
${COMPOSE_PROJECT_NAME}"
  # Start up command in the list form, which overrides the
image default command.
  entrypoint:
   - php
   -  d
   - zend_extension=/usr/local/lib/php/extensions/no-debug-
non-zts-20100525/xdebug.so
   -  d
   - memory_limit=-1
   - vendor/bin/phpunit
```

Labels

```
services:
 web:
  # Container label meta data.
  labels
   com.example.description: "Accounting webapp"
   com.example.department: "Finance"
   com.example.label-with-empty-value: ""
```

Logging

```
services:
 web:
  # Define logging.
  logging:
   driver: syslog
   options:
    syslog-address: "tcp://192.168.0.42:123"
```

Dependencies

```
services:
 web:
  build: .
  # Define startup and shutdown dependencies between
services.
  depends_on:
   - db
   - redis
 redis:
  image: redis
 db:
  image: postgres
```

Sir Wali File:

```
name: First App System
services:
 app:
  build:
   context: .
   dockerfile: Dockerfile
  ports:
   - 3000:3000
  volumes:
   - ./:/app
   - /app/node_modules
 db:
  image: mongo:latest
```

Apni File:

```
services:
 app:
  build: .
  ports:
   - "3000:3000"
  depends_on:
   - mongo
  environment:
   - MONGO_URI=mongodb://mongo:27017/mydatabase
   - DB_NAME=my_database
  networks:
   - my_own_network
 mongo:
  image: mongo
  container_name: my_mongo_container
  ports:
   - "27017:27017"     environment:
```

```
   MONGO_INITDB_DATABASE: mydatabase
  networks:
   - my_own_network
networks:
 my_own_network:  # Define a custom network for
communication between services
volumes:
 mongo-data:  # Define a named volume for MongoDB data
persistence
```

QUIZ 1 (A)

Part A: Bash Script for Docker Deployment

```
#!/bin/bash
# Pull the required Docker images
docker pull postgres:12
docker pull frontend:6.0
docker pull backend:2.1
# Run the PostgreSQL container
docker run -d --name db -p 5437:5432 postgres:12
# Run the Frontend container
docker run -d --name frontend -p 80:80 frontend:6.0
# Run the Backend container
docker run -d --name backend -p 8080:8080 backend:2.1
```

Part B: Docker Compose File

```
version: '3.8'
services:
  postgres:
    image: postgres:12
    container_name: postgres_db
    ports:
      - "5437:5432"
    volumes:
      - db_data:/var/lib/postgresql/data
  frontend:
    image: frontend:6.0
    container_name: frontend_service
    ports:
      - "80:80"
    depends_on:
      - backend
  backend:
    image: backend:2.1
    container_name: backend_service
    ports:
      - "8080:8080"
    depends_on:
      - postgres
volumes:
  db_data:
    driver: local
```

Part C: GitHub Actions CI/CD Pipeline

```
name: CI/CD Pipeline
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
jobs:
  build_and_test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3
      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '16'
      - name: Install dependencies
        run: npm install
      - name: Build the application
        run: npm run build
      - name: Run tests
        run: npm test
  deploy:
    needs: build_and_test
    if: github.event_name == 'push'
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3
      - name: Deploy Docker Compose Stack
        run: docker-compose up -d
```

Quiz#01 Section B

Solution for Question #1: Bash Script to Update config.yaml

`update_config.sh` (Bash Script):

```
#!/bin/bash
# Check if the user provided an argument
if [ $# -ne 1 ]; then
    echo "Usage: ./update_config.sh <AUTH_KEY>"
    exit 1
fi
# Store the argument in a variable
AUTH_KEY=$1

sed -i "s/REPLACE_WITH_AUTH_KEY/$AUTH_KEY/g"
 config.yaml
echo "□ Authentication key updated successfully in
 config.yaml!"
```

How to Use the Script

1. Make it executable:
2. chmod +x update_config.sh
3. Run the script with an authentication key:
4. ./update_config.sh ZXCVBNM123

□ What Happens?

□ It replaces "REPLACE_WITH_AUTH_KEY" with "ZXCVBNM123" inside config.yaml

Solution for Question #2: Docker Compose for Microservices (MySQL + Nginx)

Docker Compose File (`docker-compose.yml`)

```
version: '3.8'
networks:
  app_network:  # Define custom network
    driver: bridge
services:
  db_container:
    image: mysql:latest
    container_name: db_container
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: app_db
    ports:
      - "3306:3306"
    networks:
      - app_network
  web_container:
image: nginx:latest
    container_name: web_container
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf  # Mount nginx confi
    networks:
      - app_network
    depends_on:
      - db_container
```

Steps to Run the Containers

1. Create & navigate to the project directory:
2. mkdir microservices-app && cd microservices-app
3. Create docker-compose.yml & nginx.conf file.
4. Run the services using Docker Compose:
5. docker-compose up -d

Solution for Question #3: GitHub Actions for Node.js Project

.github/workflows/ci.yml

```
name: Node.js CI/CD Workflow
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
jobs:
  build-and-test:
    runs-on: ubuntu-latest  # Use the latest Ubuntu OS
    steps:
      - name: Checkout Repository
        uses: actions/checkout@v4  # Fetch code from GitHu
      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: "18"  # Use Node.js v18
      - name: Install Dependencies
        run: npm install  # Install project dependencies
      - name: Build Project
        run: npm run build  # Build the project
      - name: Run Tests
        run: npm run test  # Run tests to validate changes
```