1) Explore basic git and github commands

i) git --version
- This will tell the version if it's installed

- make/create a directory in vscode terminal & enter into it:
  mkdir cy20
  cd cy20

ii) git config --global username "name"

iii) git config --global user.email "email"
- These set the username & email

iv) git init
To initialize the repository within the folder

v) git status
- This specifies in which branch we are & commit
- next create a file (any file) with some content, save & execute again.

vi) git add filename
- This will add the file (a) file in indexed

vii) git commit -m "first commit"
To save this track file in our repository

- next add another line of content in the file and save it

viii) git branch branch_name
-To create a new branch

ix) git checkout branch_name
-To enter a branch

x) git checkout master

xi) git fetch
-To merge master branch & another branch

xii) git merge branch_name

xiii) git log
-To know the commit history

xiv) git branch -d branch_name
-To delete branch

xv) git checkout <commit hash>
-To go to first version of our file

- create a new repository in github & a file
- clone this new repo on the local repo
- git clone <<https link>>
- To push local repo to git hub acc
  - git remote add origin <http...>
  - git push -u origin master

2) Implement, code, build, test, configure & monitor the software app in Devops using Flask

i) sudo apt update
ii) python3 --version
iii) sudo apt install python3-pip -y
iv) pip --version
v) sudo apt install python3-venv -y
vi) mkdir flask-project
vii) cd flask-project
viii) python3 -m venv flaskenv
ix) source flaskenv/bin/activate
x) pip install flask
xi) flask --version
xii) nano app.py

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home():
    return "Hello, DevOps!"
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

xiii) python3 app.py

xiv) build:

build.sh

```
#!/bin/bash
echo "Setting up environment..."
python3 -m venv flaskenv
source ./venv/bin/activate
pip install flask
echo "Environment setup complete.
      Run the app with:
      python"
```

xv) Test:

```
import unittest
from app import app
class TestApp(unittest.TestCase):
    def test_home(self):
        tester = app.test_client()
        response = tester.get('/')
        self.assertEqual(response.
                         status_code, 200)
        self.assertEqual(response.
                         data, b"Hello, DevOps!")
if __name__ == "__main__":
    unittest.main()
```

xiv) configure:

create .env file

nano .env

flask_port = 5000

pip install python-dotenv

xv) monitor:

```
@app.route('/health')
def health():
    return {"status": "up"}, 200
```

21/2/25

3) Install and explore selenium for automated testing.

* python --version
* pip install selenium

Program:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
browser = webdriver.Firefox()
browser.get('http://www.yahoo.com')
assert 'Yahoo' in browser.title
elem = browser.find_element(By.NAME, 'p')
elem.send_keys('selenium' + Keys.RETURN)
time.sleep(10)
browser.quit()
```

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
import time

chrome_driver_path = "C:\driver\chromedriver.exe"

service = Service(executable_path=chrome_driver_path)
driver.get("https://www.google.com")
assert "Google" in driver.title
time.sleep(15)
driver.quit()
```

Unittest:

```
import unittest
from selenium import webdriver

class GoogleTestCase(unittest.TestCase):

    def setUp(self):
        self.browser = webdriver.Firefox()
        self.addCleanup(self.browser.quit)
```

```
def test_page_title(self):
    self.browser.get('http://www.
                      google.com')
    self.assertIn('Google', self.browser
                  .title)

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

4) Setting up a gradle project, understanding build scriptx (Groovy), dependency management & task automation

* Gradle ix an open source build automation tool used for java projectx.
* Supportx Groovy & Kotlin
* Performx compiling code, Running testx, creating taskx, Creating Jaa filex.

Program:

```
package gradleproj;
import java.awt.Desktop;
import java.net.URI;

public class App{
    public static void main (String[] args){
    try{
        String url = "https://www.google.
                                com";
        if (Desktop.isDesktopSupported()){
            Desktop desktop = Desktop.
                        getDesktop()
            desktop.browse(new URI
                                (url));
        }
    else{
        System.out.println("Desktop
        is not supported on this
        platform");
```

```
        }
    catch (Exception e){
        e.printStackTrace();
    }
  }
}
```

* Download gradle for windows
* Extract the zip file & paste in: Program files.
* Edit env variables & paste the path of bin file from the gradle folder.
* Go to cmd → gradle --version
  → mkdir gradleproj
  → cd gradleproject
  → gradle init
* Enter the code in build.gradle:

```
task hello {
    doLast {
        println "Hello, Gradle!"
    }
}
```

→ cmd → gradle task

→ gradle -v hello

5) Implementing Continous security with Snyk.

* Snyk is a security tool used to find & fix vulnerabilities in your code, dependencies, container image and Infrastructure as a code.

* Using Snyk website:

i) go to snyk.io

ii) Login with your Github

iii) then authorize snyk

iv) Go for your projects & you can check your project vulnerabilities under four sections C, H, M, L — Low

Critical High Moderate

* Using cmd:

1) Install Node.js

2) Type node in cmd to check the version

3) Enter ⇒ npm install -g snyk
to install snyk

4) Enter - snyk auth and grant app access.

Using Github:

1) Search for pygoat in chrome & copy the code of github & clone the repo through cmd.
2) Open the folder in VSCode.
3) Enter - snyk test
- snyk monitor

6. Develop a single containerized application using docker.

* Docker is an open source platform that enables to build, ship & run applications inside light weight & portable containers.

* It is used to package code, dependencies & run time env into a single container.

1) Provide the initial setup for docker.
2) Go to chrome & search for "github.com/docker/welcome-to-docker.git"
3) Go to cmd and create a directory
   - mkdir dockproj
   - cd dockproj
   - git clone https://github.com/docker/welcome-to-docker.git
   - cd welcome-to-docker
   - docker build -t welcome-to-docker
   - docker run -P 5000:5000 welcome-to-docker

Creating a new project:

- mkdir docker-man
- cd docker-man

- notepad app.py

app.py:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_docker():
    return "Hello Docker!"
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

requirements.txt:

```
Flask == 2.0.1
Werkzeug = 2.0.1
```

Dockerfile:

~~Dockerfile~~

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt
RUN pip install --no-cache-dir -r
requirements.txt
COPY..
```

EXPOSE 5000

CMD ["python", "app.py"]

* To convert text file to a normal file:

- cd docker-man
- ren Dockerfile.txt Dockerfile
- docker build -t docker-man .
- docker run -p 5000:5000 docker-man

8) Automate the process of running containerised applications using kubernetes.

* k8s is an open source platform for automating deployment, scaling & management of container applications.

Steps:

* create a folder in C drive,

> mkdir kub-lab

* Go to chrome & search kubernetes download for windows & open the link. copy the commands for kubectl download.

> curl.exe -LO "https://dl.k8s.io/release/v1.33.0/bin/windows/amd64/kubectl.exe"

> curl.exe -LO "https://dl.k8s.io/v1.33.0/bin/windows/amd64/kubectl.exe.sha256"

> certutil -hashfile kubectl.exe SHA256
type kubectl.exe.sha256

> $(Get-FileHash -Algorithm SHA256 .\kubectl.exe).Hash -eq $(Get-Content .\kubectl.exe.sha256)

> kubectl version --client

Minikube installation:

- Download & run the installer:

→ New -Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force

→ Invoke -WebRequest -OutFile 'c:\minikube\minikube.exe' https://github.com/kubernetes/minikube/releases/latest/download/minikube-windows-amd64.exe -UseBasicParsing

- Copy the binary path in minikube & add it to the enviroment variables
- Start your cluster

> minikube start

- Interact with your cluster

> kubectl get po -A

> minikube kubectl -- get po -A

> alias kubectl="minikube kubectl --"

> minikube dashboard

- Deploy applications:

> kubectl create deployment hello-minikube --image=kicbase/echo-server:1.0

> kubectl expose deployment hello-minikube --type=NodePort --port=8080

> kubectl get services hello-minikube

> minikube service hello-minikube

> kubectl port-forward service/hello-minikube 7080:8080

Manage your cluster

```
> minikube pause
> minikube stop
```

7) Demonstrate creating a job using Jenkins

* Jenkins is an open source automation server widely used for continuous integration & continuous delivery.
* Helps to automate building, deployment & testing of software applications.

Steps:

1. Go to chrome & type "download free jdk binaries" and open the first link.
2. Download latest LTS release.
3. Search Jenkins download & download Jenkins for windows.
4. Install, Change → set override JAVA_HOME
5. Jenkins, Set service , port 8080 or 8081
   ↓
   Local System
   Start service → disable, install
6. Set env variable, Path → edit → new
   Go to C:drive,
   program files → Eclipse Adoptium
   → jdk 21 → bin
7. Go to services, Right click on Jenkins & click start.

8. Go to chrome & type localhost:8081:

9) Configuration management with Ansible:
Inventory, playbooks, modules, automating server configuration with playbooks.

* Ansible is an open source tool used for configuration management, appln deployment, task automation & orchestration of multinode system.

Basic components:
- Inventory → List of hosts
- modules → Reusable units of work
- Playbooks → YAML file
- Roles → way to organise playbooks & tasks into reusable comps.

Steps:
Go to terminal in Ubuntu
> sudo apt update
> sudo apt install ansible
> ansible --version
> nano hosts.ini
[local] → Group name
localhost ansible_connection=local
Save the file

> nano setup.yml

```
---
- name : Basic Server Setup
  hosts : local
  become : yes
  tasks :
    - name : Update apt cache
      apt :
        update_cache : yes
    - name : Install curl
      apt :
        name : curl
        state : present
```

* Save the file
* To execute playbook =>

> sudo ansible-playbook -i hosts.ini setup.yml

10) Creating a maven project, understanding pom file, dependency management & plugins.

* Maven is an open-source build automation & management tool mainly used for java based project.
* It simplifies the build process, dependency management, reporting & documentation.
* pom.xml contains project metadata, dependencies, plugins, build conf, ...

Steps:

* Go to terminal in Ubuntu:
  - > sudo apt update
  - > sudo apt install openjdk-11-jre -headless -y
  - > sudo apt install maven -y
  - > sudo apt install git
* Create a new directory
  - > mkdir maven
  - > cd maven

> git clone https://github.com/krish
pluto/BoardgamelistingWebApp.git

~~Execute maven~~

> cd BoardgamelistingWebApp

Execute maven commands:

>mvn clean
>mvn validate
>mvn test
>mvn compile
>mvn package

* Go to target file
>cd target
>java -jar database-service-
project-0.0.1.jar

* Go to chrome and search local
host:8080.