|  |  |
| --- | --- |
| **EXC 1** | **Git Commands & GitHub** |
| **10/07/2024** |

**AIM: Exploring Git Commands through Collaborative Coding – Basic Git commands.**

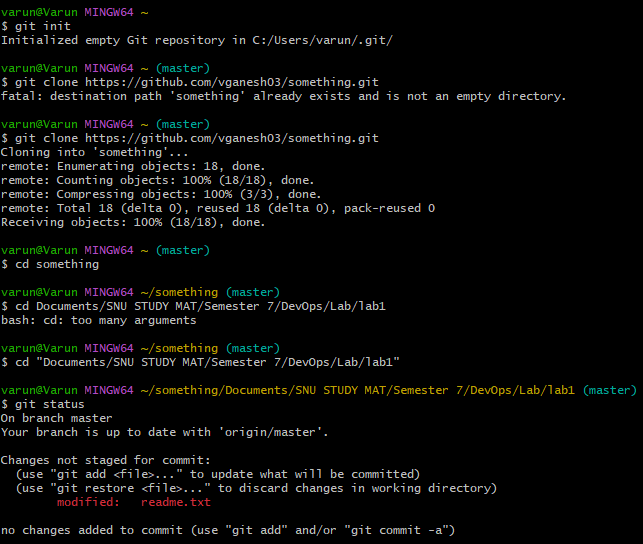
**COMMANDS & SCREENSHOTS:**

A computer screen shot of text

Description automatically generated

A computer screen shot of a black screen

Description automatically generated



A computer screen with yellow text

Description automatically generated

A computer screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

**RESULT:** Thus, the following commands were successfully implemented.

|  |  |
| --- | --- |
| **EXC 2** | **Git Commands & GitLab** |
| **06/08/2024** |

**AIM: Implement GitLab Operations using Git**

**SCREENSHOTS & OUTPUT:**

A computer screen shot of a program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screenshot of a chat

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a program

Description automatically generated

**RESULT:** Thus, the git commands were successfully implemented with GitLab.

|  |  |
| --- | --- |
| **EXC 3** | **Git Commands & BitBucket** |
| **06/08/2024** |

**AIM:** Implement Git commands with BitBucket.

**SCREENSHOTS & OUTPUT:**

A black screen with white text

Description automatically generated

A computer screen with white text

Description automatically generated

A computer screen with text

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screenshot of a computer

Description automatically generated

A close-up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

**RESULT:** Thus, the git commands were successfully implemented with BitBucket.

|  |  |
| --- | --- |
| **EXC 4** | **Git Advanced Commands** |
| **08/08/2024** |

**AIM:** Implement the git advanced commands

**SCREENSHOTS & OUTPUT:**

Rebase:

A computer screen shot of text

Description automatically generated

A computer screen shot of a program

Description automatically generated

A screenshot of a computer

Description automatically generated

Stash:

A screen shot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generatedRevert:

Reset:

A computer screen shot of text

Description automatically generated

Cherry-Pick

A computer screen shot of text

Description automatically generated

A computer screen shot of text

Description automatically generated

Sub Module:

A computer screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**RESULT:** Thus, the advanced git commands were implemented successfully.

|  |  |
| --- | --- |
| **EXC 5** | **Maven POM** |
| **28/08/2024** |

**AIM:** Install Maven and examine the POM File

**SCREENSHOTS & OUTPUT:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generatedA computer screen with white text

Description automatically generated

**RESULT:** Maven was installed and POM File was examined successfully.

|  |  |
| --- | --- |
| **EXC 6** | **JAR File** |
| **28/08/2024** |

**AIM:** Initiate a build execution.

**SCREENSHOTS & OUTPUT:**

A screenshot of a computer

Description automatically generated

**RESULT:** Build was executed successfully.

|  |  |
| --- | --- |
| **EXC 7** | **Maven Questionnaire** |
| **13/09/2024** |

**AIM:** Solve the Questionnaire pertaining to Maven

**SOLUTIONS:**

Question 1: Customizing Maven Build Lifecycle

Answer: 1. The changes to be made in pom.xml are: In the <build> section of your pom.xml, we can configure the maven-surefire-plugin (used for running unit tests) and bind it to the integration-test phase.

Question 2: Adding External Dependencies and Managing Versions

Answer 2: To ensure consistent Spring Framework versions across a multi-module project, we can use the dependencyManagement section in the parent pom.xml. This ensures that all modules use the same version of dependencies.

Question 3: Skipping Tests During Build

Answer 3. “mvn clean install -DskipTests” this command skips all unit and integration testing.

Question 4: Managing Plugins for Code Coverage

Answer 4. To integrate JaCoCo for code coverage, and ensure the build fails if coverage is below 80%, we can modify the pom.xml to add the JaCoCo plugin and configure the rule to break the build.

Question 5: Managing Multi-Module Project Dependencies

Answer 5. In a multi-module project, if Module A depends on Module B, and we want Module A to automatically reflect changes from Module B, we need to structure the POMs to specify Module B as a dependency of Module A.

Question 6: Overriding a Dependency Version in a Child Module

Answer 6. If a child module needs a different version of a dependency (e.g., Hibernate) than what is defined in the parent POM, you can override the version in the child module’s pom.xml.

Question 7: Using Maven Properties to Manage Project Information

Answer 7. We can define properties within the <properties> tag of the pom.xml. We can then use these properties elsewhere in the pom.xml by referencing them with ${}

Question 8: Enforcing a Minimum Java Version for the Project

Answer 8. We can configure the maven-compiler-plugin in the pom.xml. both the source and target Java versions are set to 11, ensures the project only builds with Java 11 or higher.

Question 9: Customizing the Default Build Lifecycle

Answer 9. we can configure the Checkstyle plugin and bind it to the validate or compile phase. ensures that static code analysis is performed before the test phase in the build lifecycle.

Question 10: Skipping Specific Lifecycle Phases

Answer 10. “mvn verify -DskipTests”, This command will skip the unit tests (in the test phase) but continue with the integration-test and verify phases.

Question 11: Binding Goals to Custom Phases

Answer 11. we can bind it to a custom Maven profile. Profiles allow us to specify environment-specific configurations in Maven. Configure flyway plugin, bind it to custom phase, define profiles and run this command: “mvn clean install -Pstaging -Denv=staging”, ensures that Flyway runs only in the appropriate environments, and we can switch between environments using Maven profiles.

Question 12: Handling Failed Builds

Answer 12. We can define profiles in pom.xml, allow us to define different build configurations for different environments, such as development, testing, and production. Using this command: “mvn clean install -Pdevelopment”, to activate a profile.

Question 13: Using the Maven Site Lifecycle

Answer 13. configure the Maven Surefire plugin and bind it to the testing profile to ensure unit tests only run when the testing profile is active.

Question 14: Multi-Module Build with Independent Modules

Answer 14. add a custom goal, such as generating API documentation with Javadoc, we can bind the Javadoc plugin to a specific phase like package or install. So configure Javadoc plugin in pom.xml.

Question 15: Customizing the Clean Lifecycle

Answer 15: We can customize the build output directories (such as where compiled classes and resources are stored) using the <build> section of the pom.xml. allows us to organize our project’s artifacts (classes, resources) in custom directories, depending on our project structure or needs.

Question 16: Using Plugin Inheritance with Custom Execution Goals

Answer 16: Managing different versions of the same library, known as dependency version conflicts, is a common issue in Maven. By using <exclusions> tag, using dependency management, maven enforcer plugin, ensures that the project adheres to the specified versions and avoids using different versions of the same library.

**RESULT:** The above questionnaire was answered successfully.

|  |  |
| --- | --- |
| **EXC 8** | **Jenkins & Webhooks** |
| **21/10/2024** |

**AIM:** Applying CI/CD Principles to Web Development Using Jenkins, Git, and Local HTTP Server

**PROCEDURE:**

1. Setup a simple webapp using ExpressJS.

2. create a git repository and commit/push.

3. setup a webhook in github with the help of ngrok.

4. create a Jenkins pipeline, perform the windows batch commands in “build”.

5. record output.

**SCREENSHOTS & OUTPUTS**

A computer screen shot of a program code

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screenshot of a webbook

Description automatically generated

**RESULT:** Thus, CI/CD Principles with Jenkins & Webhooks were implemented successfully.

|  |  |
| --- | --- |
| **EXC 9** | **Docker** |
| **23/10/2024** |

**AIM:** Exploring Containerization and Application Deployment with Docker

**PROCEDURE:**

1. Install docker
2. Create simple webpage
3. Create dockerfile
4. Build docker image
5. Run docker container
6. Cleanup

A screen shot of a computer

Description automatically generated**SCREENSHOTS & OUTPUT:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**RESULT:** Thus, the docker image was run successfully.

|  |  |
| --- | --- |
| **EXC 10** | **Jenkins & Docker** |
| **24/10/2024** |

**AIM:** Applying CI/CD Principles to Web Development Using Jenkins, Git, using

Docker Containers

**PROCEDURE:**

1. Setup webapp using ExpressJS, containerize using Docker

2. Setup github repo, commit and push

3. Open Jenkins, create job

4. Add parameters, github\_repo, branch specifier, build triggers.

5. Configure build settings.

6. Trigger the CI/CD pipeline and check output

**OUTPUTS & SCREENSHOTS:**

A computer code on a black background

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a video game

Description automatically generated

A screenshot of a webbook

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**RESULT:** Thus, CI/CD Principles with Jenkins, Docker and Git were implemented successfully.

|  |  |
| --- | --- |
| **EXC 11** | **Jenkins & NodeJS** |
| **24/10/2024** |

**AIM:** Applying CI/CD Principles for NodeJS application Using Jenkins, Git, using Docker Containers

**PROCEDURE:**

1. Create **GitHub Repository**: Go to GitHub, click "New repository," name it (e.g., devops\_11), and create it.
2. Create **Node.js Application**: Initialize a Node.js project with npm init -y and create app.js with an Express server.
3. Create **Dockerfile**: Create a Dockerfile that sets up the Node.js environment, installs dependencies, and runs app.js.
4. Configure **Port Numbers**: Ensure EXPOSE 3000 in the Dockerfile matches the port in app.js, and plan to run the container on a free port (e.g., 3001).
5. Create **GitHub Personal Access Token (PAT)**: Generate a PAT in GitHub settings under Developer settings and copy it.
6. Push **Code to GitHub**: Initialize Git, add files, commit changes, and push to your GitHub repository.
7. Set **Up Jenkins Pipeline**: Create a new pipeline job in Jenkins with a Groovy script to clone the repo, build the Docker image, and deploy it.
8. Run **the Jenkins Pipeline**: Trigger the pipeline in Jenkins and monitor the console output for success or errors.

A screenshot of a computer

Description automatically generated**OUTPUTS & SCREENSHOTS:**

A computer screen shot of a program code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A close-up of a message

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

**RESULT:** Thus, CI/CD Principles with Jenkins, Git, NodeJS were implemented successfully.

|  |  |
| --- | --- |
| **EXC 12** | **Blue-Green Deployment** |
| **24/10/2024** |

**AIM:** Implement a Blue-Green Deployment strategy for a Node.js application.

**PROCEDURE:**

1. Setup webapp in express.js and containerize it.
2. Create a dockerfile
3. Create github repository and commit/push
4. Create Jenkins job
5. Select pipeline, and add the following jenkinsfile groovy script:  
   pipeline {

agent any

environment {

DOCKER\_HUB\_REPO = 'vganesh03/hello-world-app'

BLUE\_ENV = 'blue\_env'

GREEN\_ENV = 'green\_env'

}

stages {

stage('Build') {

steps {

script {

docker.build("${DOCKER\_HUB\_REPO}:${env.BUILD\_ID}")

}

}

}

stage('Deploy to Blue') {

steps {

script {

def blueContainer = docker.image("${DOCKER\_HUB\_REPO}:${env.BUILD\_ID}")

// Use double quotes to interpolate the variable

blueContainer.run("-d -p 3001:3000 --name ${env.BLUE\_ENV}")

}

}

}

stage('Test Blue') {

steps {

script {

echo 'Running tests on Blue environment...'

}

}

}

stage('Deploy to Green') {

steps {

script {

def greenContainer = docker.image("${DOCKER\_HUB\_REPO}:${env.BUILD\_ID}")

// Use double quotes to interpolate the variable

greenContainer.run("-d -p 3002:3000 --name ${env.GREEN\_ENV}")

}

}

}

stage('Test Green') {

steps {

script {

echo 'Running tests on Green environment...'

}

}

}

stage('Cleanup') {

steps {

script {

// Use double quotes to interpolate the variable

sh "docker rm -f ${env.BLUE\_ENV} || true"

sh "docker rm -f ${env.GREEN\_ENV} || true"

}

}

}

}

}

1. Check for output.

**OUTPUTS & SCREENSHOTS:**

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated





**RESULT:** Blue-green deployment was successfully implemented.