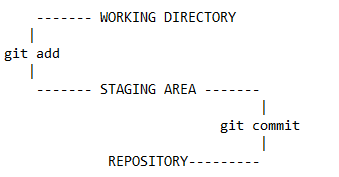
**Git Architecture**

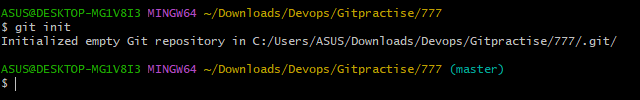


* Working Directory - a space where you makes changes to a file/directory
* After using git add, the file moves to Staging area but “.git” does not capture any information of the file as it is still not moved to the repository
* After using git commit, the file is in a repository and .git captures all the information of the file

**Git commands:**

1. Initializing git:

git init

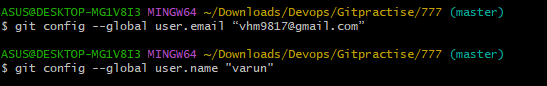


**Configuring username and mail Id**

1. Set the name the mail/name that will attach to your commits

git config --global user.email “mail\_id”

git config --global user.name “username”



**Getting and creating new projects**

1. Creating and initializing new project

git init <new\_project\_name> 

1. Cloning a repository using https

git clone [URL]



1. Cloning a repository using SSH

Steps

1. Open git bash and generate ssh keys/rsa keys/public and private keys by using

ssh-keygen (or) ssh-keygen -t rsa

Note:

1. ssh-keygen by itself generates an Ed25519 key, which is generally recommended for its security and performance.
2. The ssh-keygen -t rsa command generates an RSA key, which may be necessary for compatibility with older systems
3. view the path of the public key to copy the public key

cat /c/Users/ASUS/.ssh/id\_ed25519.pub

Note: here in case of ASUS if there is "varun hm" then key will not be shown as there is space in between, so use, "varun\hm" in place of "varun hm" and run the command and copy the public key.

1. Go to Github, go to profile on the right top ---> settings ---> SSH & GPG keys (left side)

--> add tile name and paste the copied public key(from local) in the 'key section' ----> click on add

d. Go to the respective repository ---> code ---> ssh ---> copy the key

e. Go back to local,

git clone <paste the copied ssh key>

Now in local,

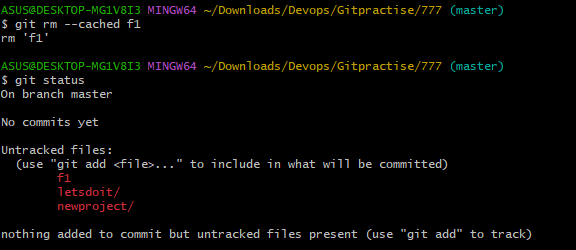
repository is created/synced from GitHub and it is already already initialized since we are pulling files from Centralized repository(GitHub) and all files in github are visible in the local repo.

**BASIC SNAPSHOTTING:**

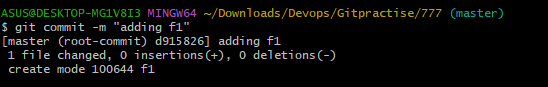
1. Adding files to staging area:
   1. git add <file>
   2. git add . ------ to stage all changes
   3. git add f1 f2 f4 --------- to stage multiple changes of multiple files



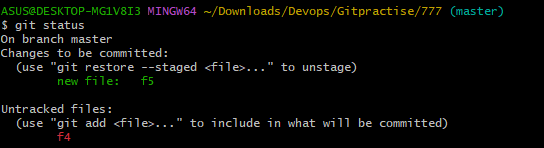
1. git rm --cached <file> ----- to unstage a file from staging area



1. git commit –m “message” ------- to commit a file from staging area (coming the changes

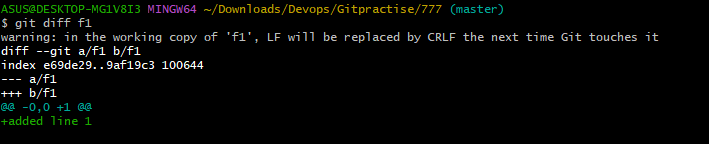


1. git status ----- displays status of changes as untracked, modified or staged.



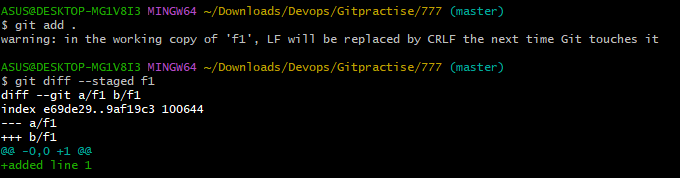
1. git diff ------ shows difference between working directory and index(staging area)

git diff <filename> ---- for a particular file

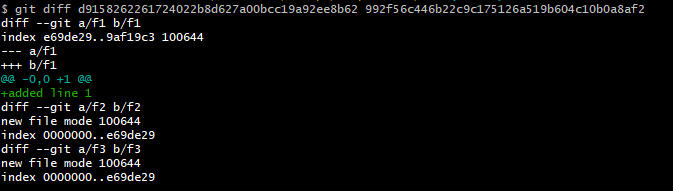


git diff –staged ----- shows difference between index and last commit

git diff –staged <filename> ---- for a particular file

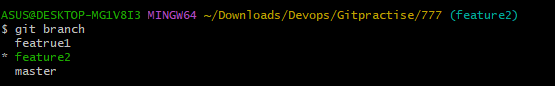


git diff [commit1] [commit2] ------ shows difference between 2 commits



**Branching and Merging**

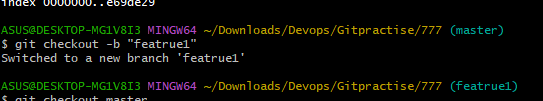
1. git branch --- lists all branch



1. git branch [new\_branch\_name] ------- to create new branch

git branch checkout [branch\_name] ---- to switch to a specific branch

git branch checkout –b [new\_branch\_name] ---- to create and switch to new branch



1. git branch –d [branch\_name] ---- to delete a branch

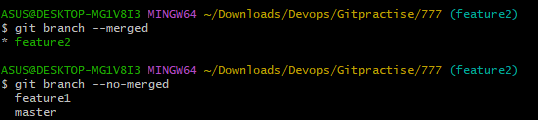


1. git merge [branch\_name] --- to merge a branch



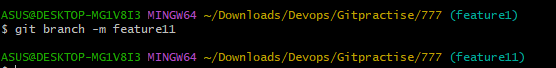
1. git branch --no-merged ----- to display unmerged branch w.r.t current branch

git branch --merged ----- to display merged branch w.r.t current branch

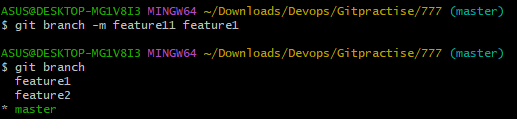


**Renaming Branch**

1. git branch –m [new\_name] ---- must be in the branch you want to rename

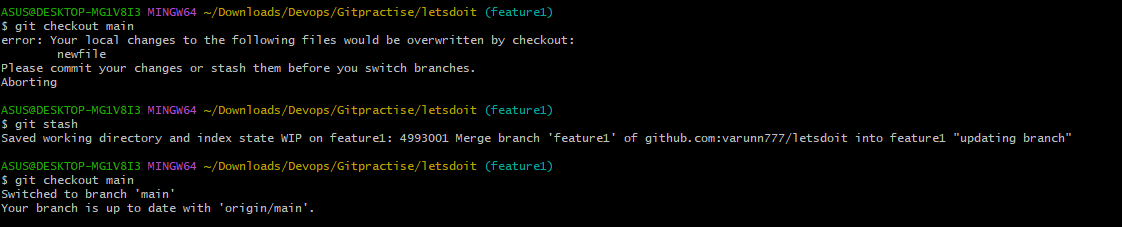


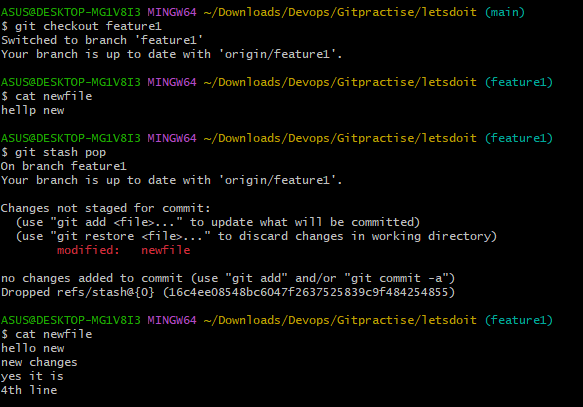
1. git branch –m [old\_name] [new\_name] --- when you are in master/main



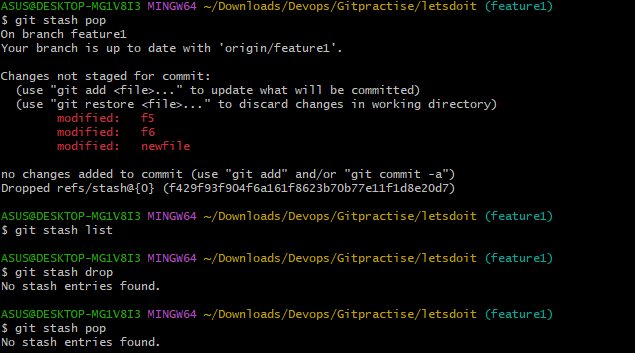
**Stashing**

Git stash 🡪 save changes temporarily in an inode storage

  
git stash pop 🡪 apply most recently popped changes

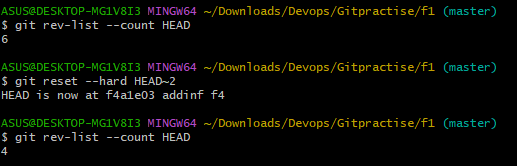


git stash drop 🡪 drops/removes all stashed messages

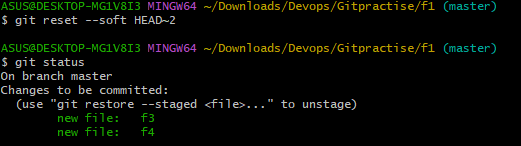


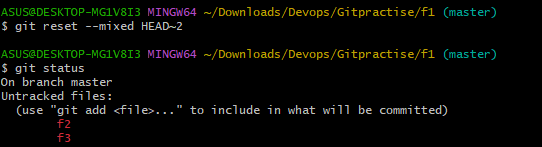
**Git reset (3types)**

git reset –hard HEAD~2 --🡪 removes/deletes 2 recent commits completely from the history



git reset –soft HEAD~2 🡪 moves the last 2 commits to staging area



git reset –mixed HEAD~2 🡪 moves the last 2 commits to working directory

**Git Blame** - git blame is a command in Git used to display the last modification for each line in a file. This command helps you identify who made changes to the code and when those changes were made.

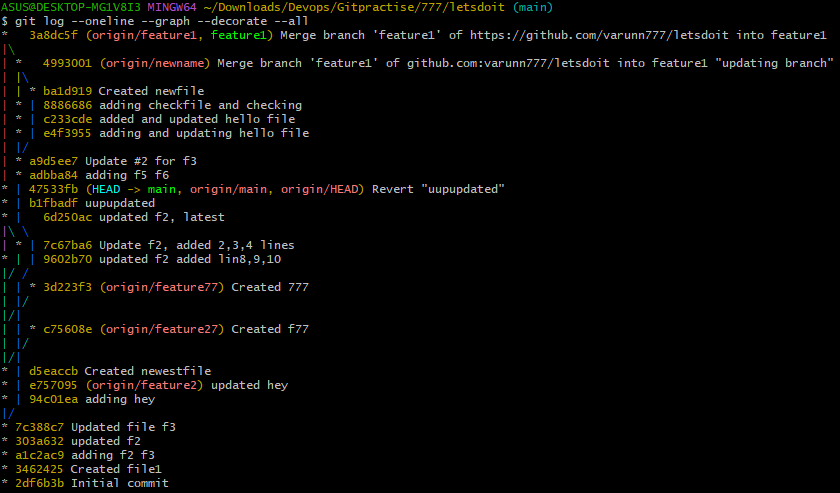
git blame filename

**Inspection and Comparison**

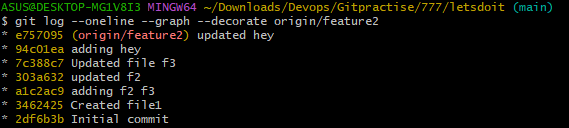
1. git log ---- shows commit history



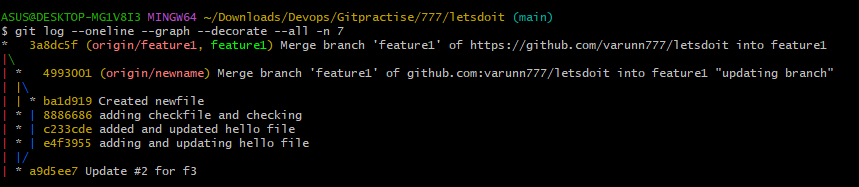
1. git log --oneline --graph --decorate --all --------- shows graph for commit history



1. git log --oneline --graph --decorate <branchname> -------- graph for a specific branch



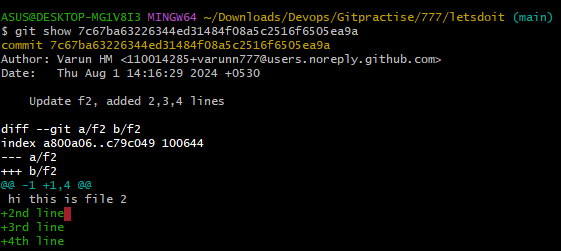
1. git log --oneline --graph --decorate --all –n 7 ---------limits the no. of commit to show in graph



1. git log --oneline --graph --decorate --all -- filename

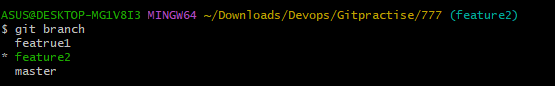


1. git show -------- shows information about a specific commit



**Tagging commits**

1. git tag <tag-name> ---> creates new tag for the recent commit

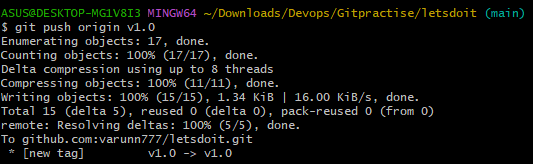
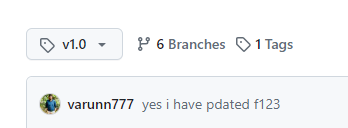




1. git tag -d <tag-name> ---> delete a tag



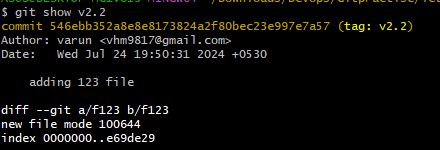
1. push origin v1.0 ---> push a tag to a remote repository



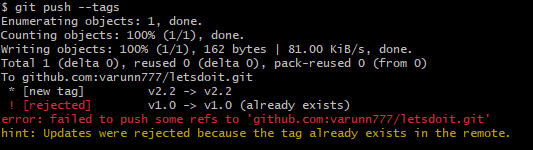
1. git tag 🡪 to list tags



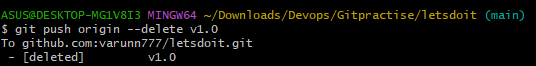
1. show <tagname> -🡪 to display tag with details



1. git push --tags 🡪 to push all tags to remote repo

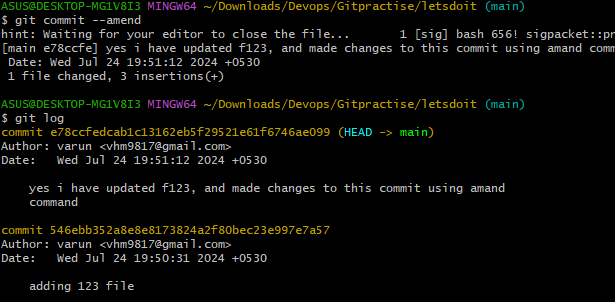


1. git push origin --delete tagname -🡪 to delete a tag remotely



**Rewriting History**

1. git commit --amend --🡪 modify’s the most recent commit or the last commit

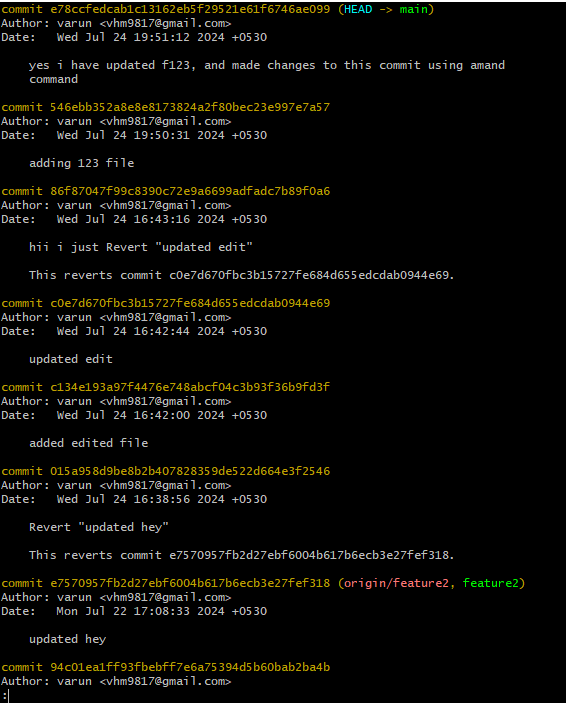


1. git rabase -i [base]

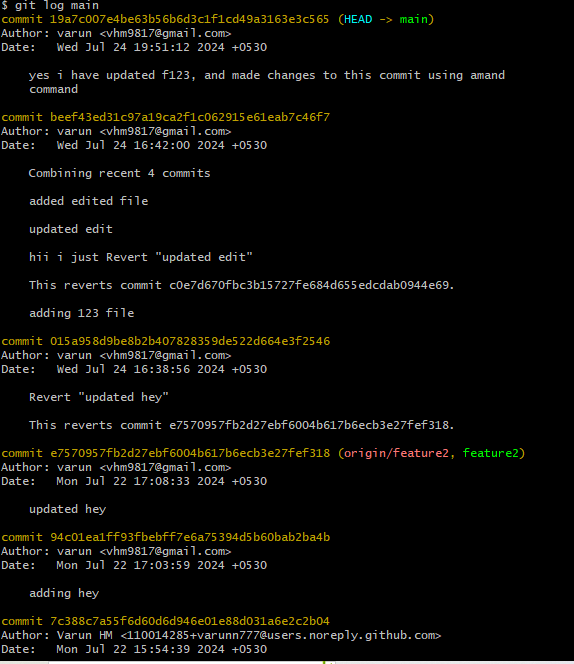
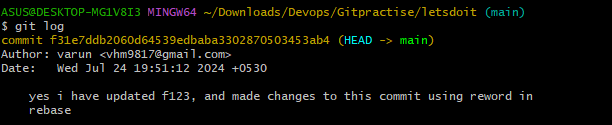
git rebase --i HEAD~8

1. pick -> use commit as it is
2. squash 🡪combine this commit with other commits and edit the commit message
3. edit 🡪 use commit, but stop for amending
4. reword 🡪 use commit, but edit commit message
5. fixup 🡪 combine this commit with previous commit, keeing the previous commit changes

Before:-

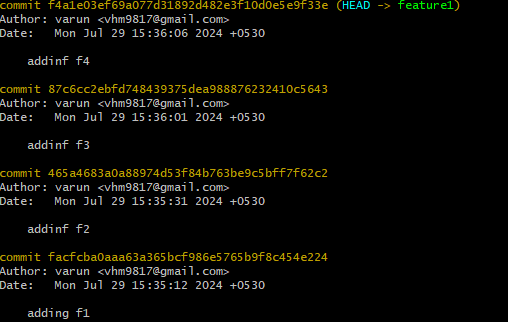
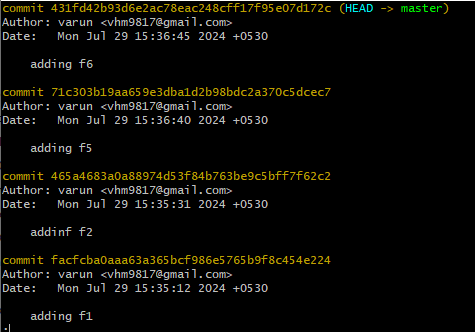


After:

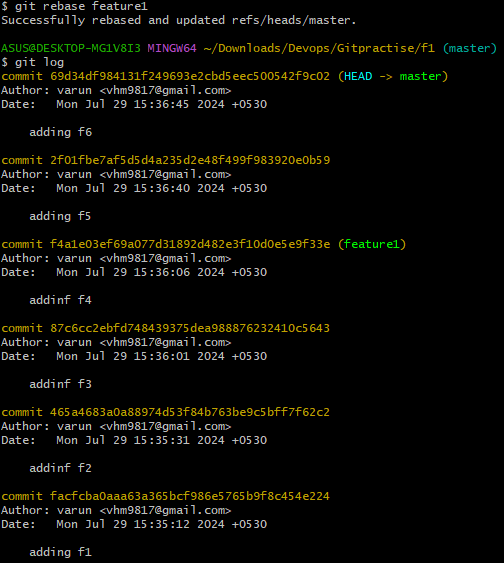


**Advance Branching and Merging**

1. git rebase branchname



git rebase feature1



using "git rebase"

* + create and commit files f1 and f2 seperately in main
  + switch to feature branch and commit 2 new files f3 and f4 seperately
  + switch back to master and commit 2 new files f5 and f6 seperately
  + now, using "git rebase feature"

observation:-

1. no message popped out/ no editor opened

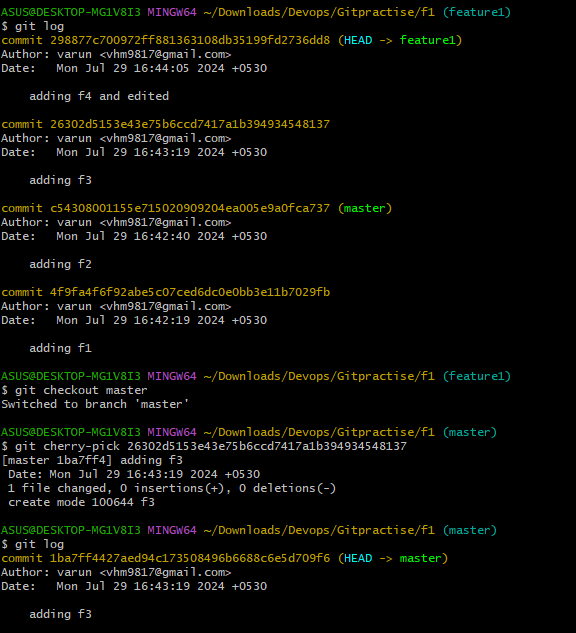
2. no extra commit added

3. the branches are mereged but the commit ids for f5 and f6 has been changed.

4. the order remains the same as the order of creation of commits

2. git cherry-pick

Git cherry-pick <commit\_id/commit\_hash> 🡪 allows you to apply the changes introduced by some existing commits onto your current branch. This is particularly useful when you want to port specific commits from one branch to another. (commit\_ids will be same)



**Git, Maven, Java, Tomcat installation and installing a java project – Automation**

#!/bin/bash

# Update package manager

echo "Updating package manager..."

sudo yum update -y

# Install git

echo "Installing git..."

sudo yum install git -y

# Install Maven

echo "Installing Maven..."

sudo yum install maven -y

# Check if Java is installed

echo "Checking if Java is installed..."

if ! java -version &> /dev/null; then

echo "Java is not installed. Installing Java..."

sudo yum install -y java-1.8.0-openjdk

else

echo "Java is installed"

fi

# Verify Java installation

java -version

# Define Tomcat version and URL

TOMCAT\_VERSION=9.0.93

TOMCAT\_URL=https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.93/bin/apache-tomcat-9.0.93.tar.gz

# Download Tomcat

echo "Downloading Tomcat..."

wget $TOMCAT\_URL

# Extract Tomcat

echo "Extracting Tomcat..."

tar -xvzf apache-tomcat-${TOMCAT\_VERSION}.tar.gz

# Remove the tar.gz file

echo "Removing tar.gz file..."

rm -rf apache-tomcat-${TOMCAT\_VERSION}.tar.gz

# Rename the Tomcat directory

echo "Renaming Tomcat directory..."

mv apache-tomcat-${TOMCAT\_VERSION} /home/ec2-user/tomcat

# Define variables

TOMCAT\_HOME="/home/ec2-user/tomcat"

MANAGER\_CONTEXT="$TOMCAT\_HOME/webapps/manager/META-INF/context.xml"

HOST\_MANAGER\_CONTEXT="$TOMCAT\_HOME/webapps/host-manager/META-INF/context.xml"

TOMCAT\_USERS="$TOMCAT\_HOME/conf/tomcat-users.xml"

# Function to modify context.xml files

modify\_context\_xml() {

local context\_file=$1

if grep -q 'Valve className="org.apache.catalina.valves.RemoteAddrValve"' "$context\_file"; then

echo "Updating $context\_file..."

sudo sed -i 's|<Valve className="org.apache.catalina.valves.RemoteAddrValve".\*|<!-- <Valve className="org.apache.catalina.valves.RemoteAddrValve\" allow=\"127\\.\\d+\\.\\d+\\.\\d+|::1\" /> -->|' "$context\_file"

else

echo "No matching Valve class found in $context\_file. No changes made."

fi

}

# Function to modify tomcat-users.xml file

modify\_tomcat\_users() {

local users\_xml=$1

local roles\_and\_user='

<role rolename="manager-gui"/>

<role rolename="manager-script"/>

<role rolename="manager-jmx"/>

<role rolename="manager-status"/>

<user username="tomcat" password="tomcat123" roles="manager-gui,manager-script,manager-jmx,manager-status"/>'

if grep -q '<user username="tomcat"' "$users\_xml"; then

echo "User tomcat already exists in $users\_xml."

else

echo "Adding roles and user to $users\_xml..."

sudo sed -i "/<\/tomcat-users>/ i\ $roles\_and\_user" "$users\_xml"

fi

}

# Modify context.xml files for manager and host-manager

echo "Modifying context.xml files for manager and host-manager..."

modify\_context\_xml "$MANAGER\_CONTEXT"

modify\_context\_xml "$HOST\_MANAGER\_CONTEXT"

# Modify tomcat-users.xml file to add roles and user

echo "Modifying tomcat-users.xml file to add roles and user..."

modify\_tomcat\_users "$TOMCAT\_USERS"

# Cloning project from repository

echo "Cloning project from repository..."

git clone https://github.com/varunn777/java-hello-world-with-maven.git

# Changing the directory

cd java-hello-world-with-maven

# Running all the Maven commands, skipping integration testing and deploy

echo "Running Maven commands..."

mvn validate

mvn compile

mvn test

mvn package

mvn install

# Copying the JAR file from the target folder to Tomcat's webapps directory

echo "Copying JAR file to Tomcat's webapps directory..."

cp target/jb-hello-world-maven-0.2.0.jar "$TOMCAT\_HOME/webapps/"

# Start Tomcat to apply changes

echo "Starting Tomcat..."

"$TOMCAT\_HOME/bin/startup.sh"

echo "Tomcat configuration updated and started successfully."