**Git Assignment solutions**

Assignment1: ● Create a new folder

● Put the following files in the folder

○ Code.txt

○ Log.txt

○ Output.txt

● Stage the Code.txt and Output.txt files

● Commit them

● And finally push them to github

Assignment1 solution:

mkdir gittask1

cd gittask1

git init

touch master.txt

la

git add . && git commit –m “Master branch generation”

git status

git branch

touch Code.txt

touch Log.txt

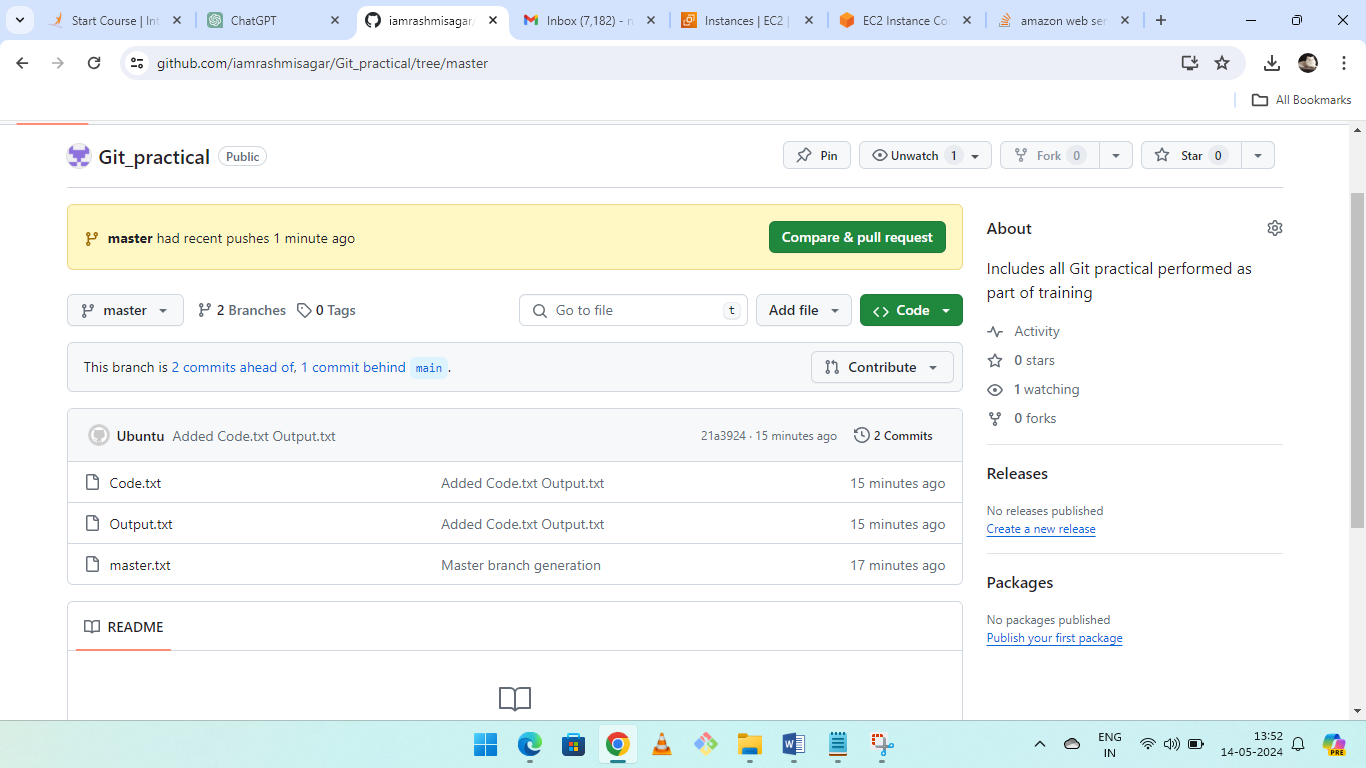
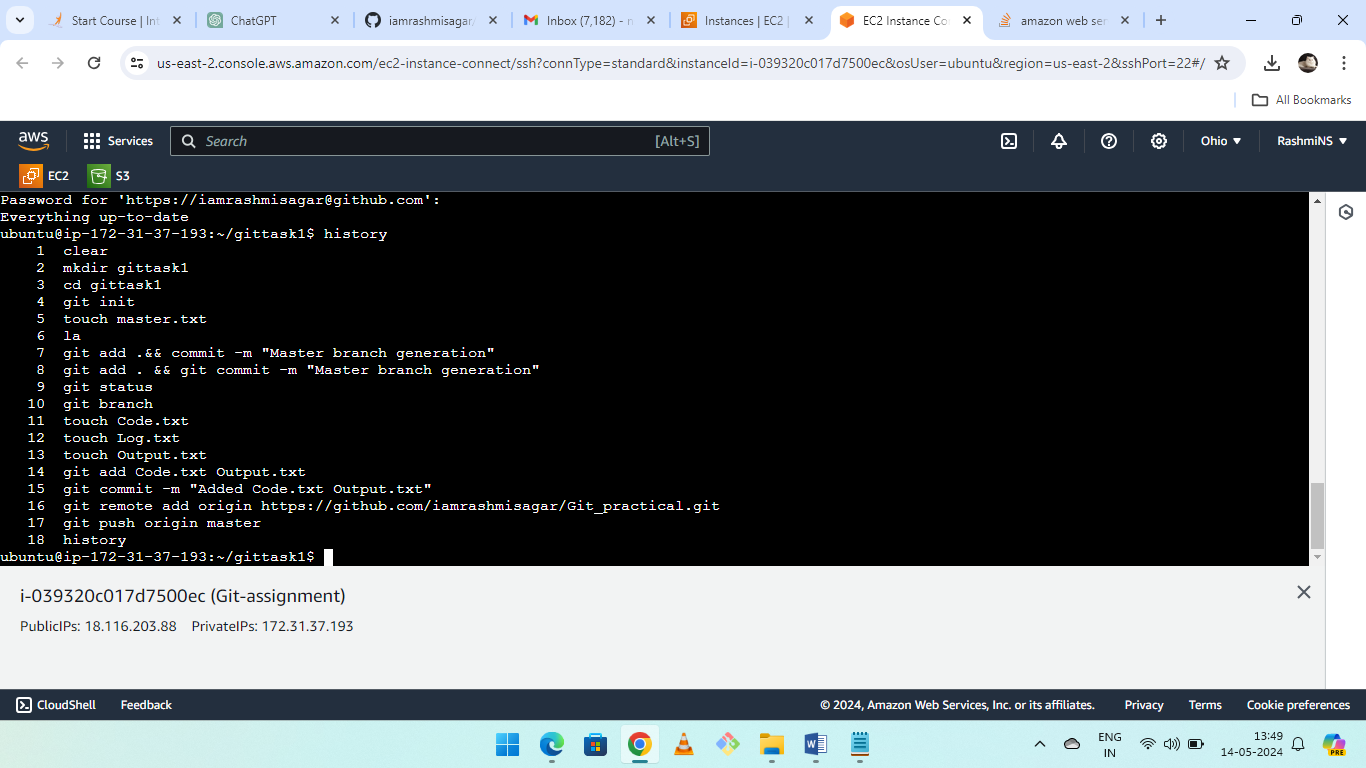
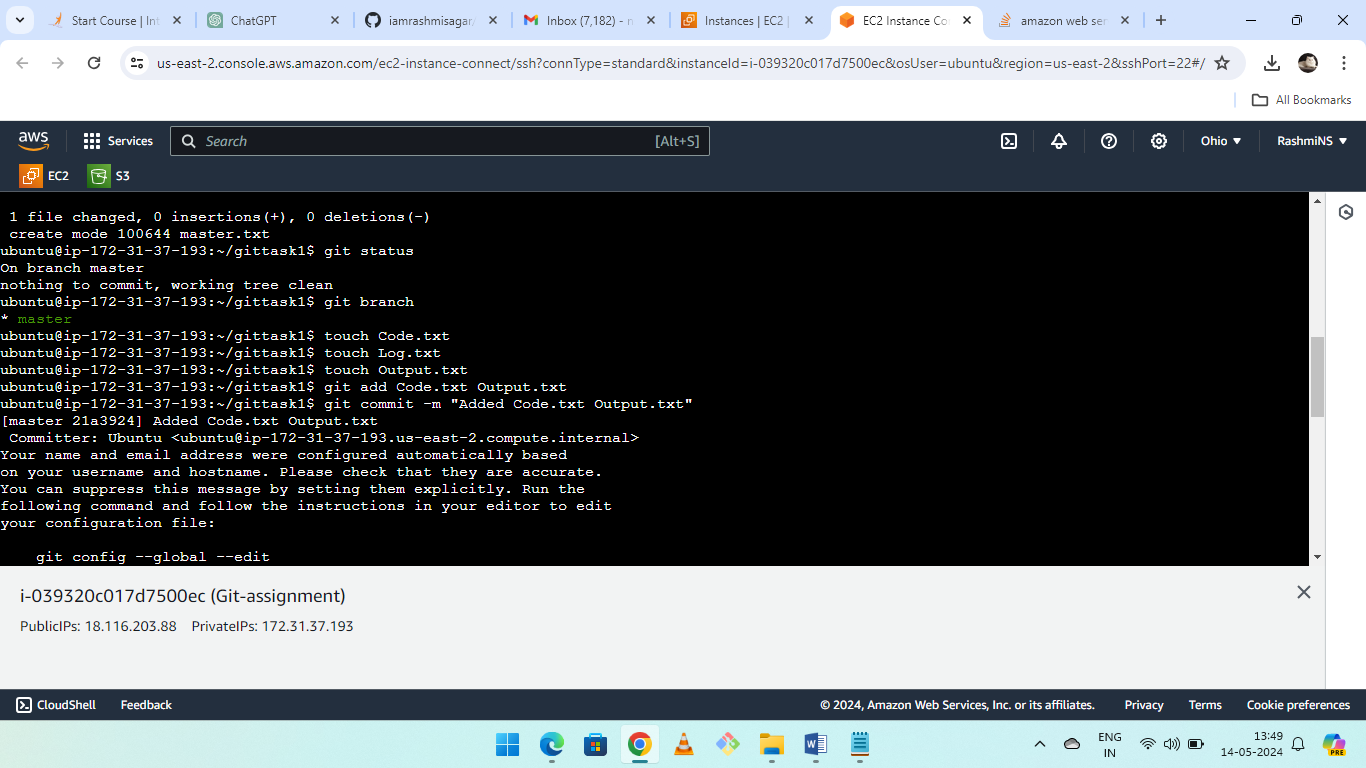
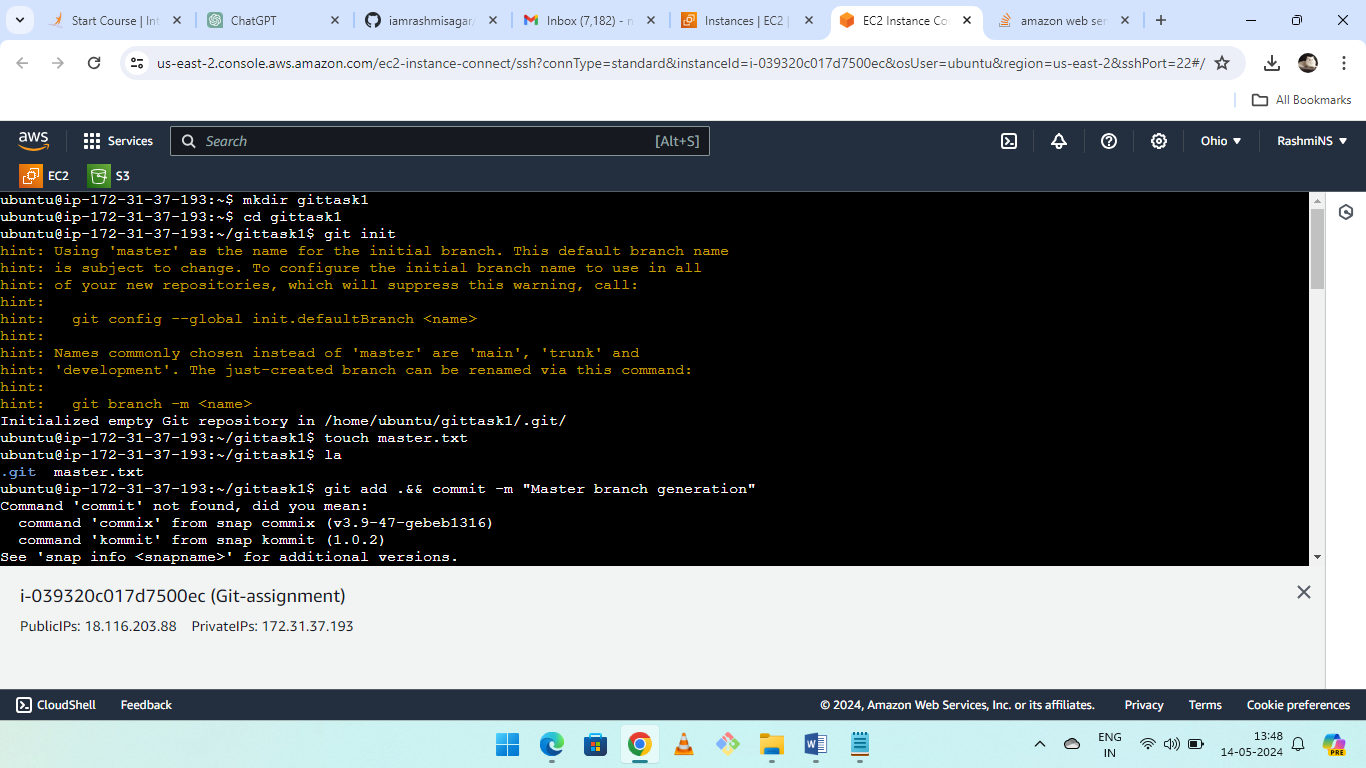
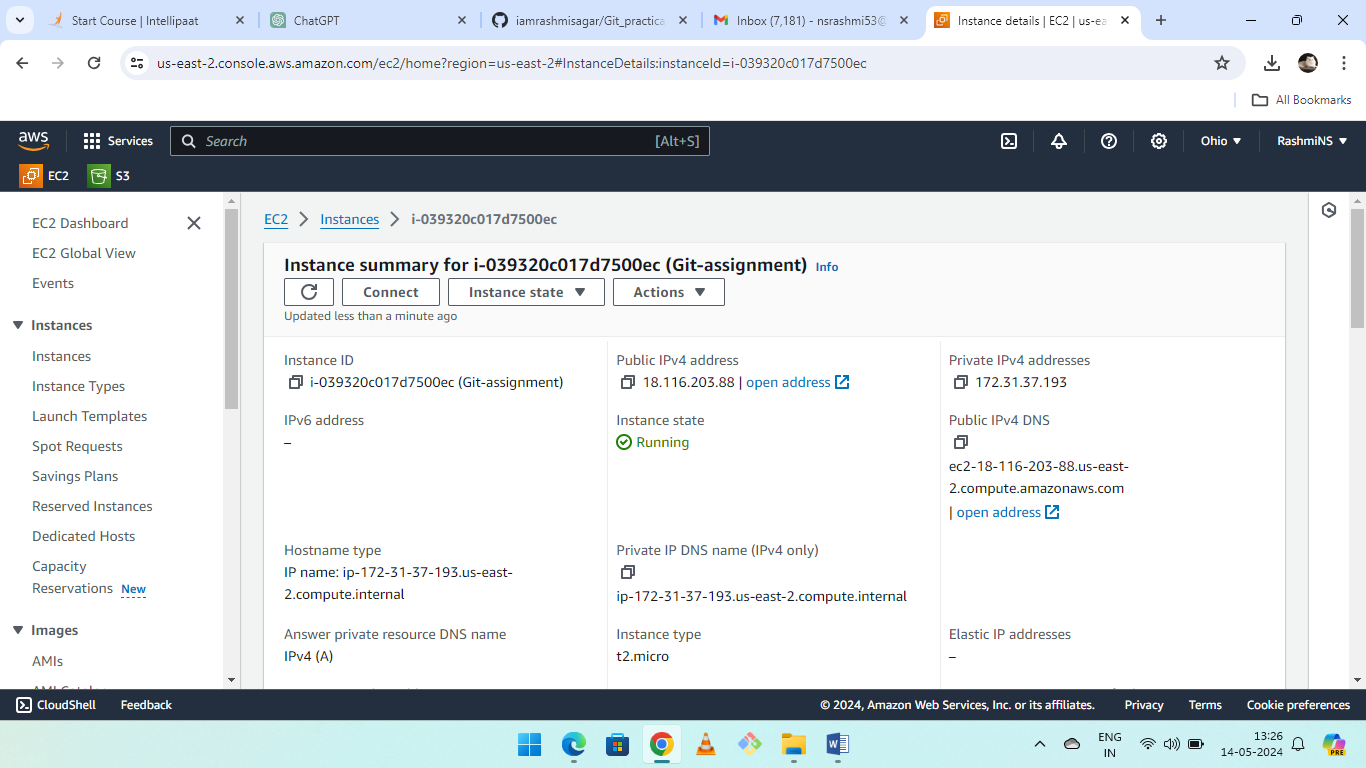
touch Output.txt

git add Code.txt Output.txt

git commit -m "Added Code.txt and Output.txt"

git remote add origin <https://github.com/iamrashmisagar/Git_practical.git>

git push origin master



Assignment2: Create a git working directory with feature1.txt and feature2.txt in the master branch

● Create 3 branches develop, feature1 and feature2

● In develop branch create develop.txt, do not stage or commit it

● Stash this file, and checkout to feature1 branch

● Create new.txt file in feature1 branch, stage and commit this file

● Checkout to develop, unstash this file and commit

**Assignment2 solution**:

cd gittask1

git init

touch feature1.txt feature2.txt

git add .

git commit -m "Initial commit with feature1.txt and feature2.txt"

git branch develop

git branch feature1

git branch feature2

git checkout develop

touch develop.txt

git stash

git checkout feature1

touch new.txt

git add new.txt

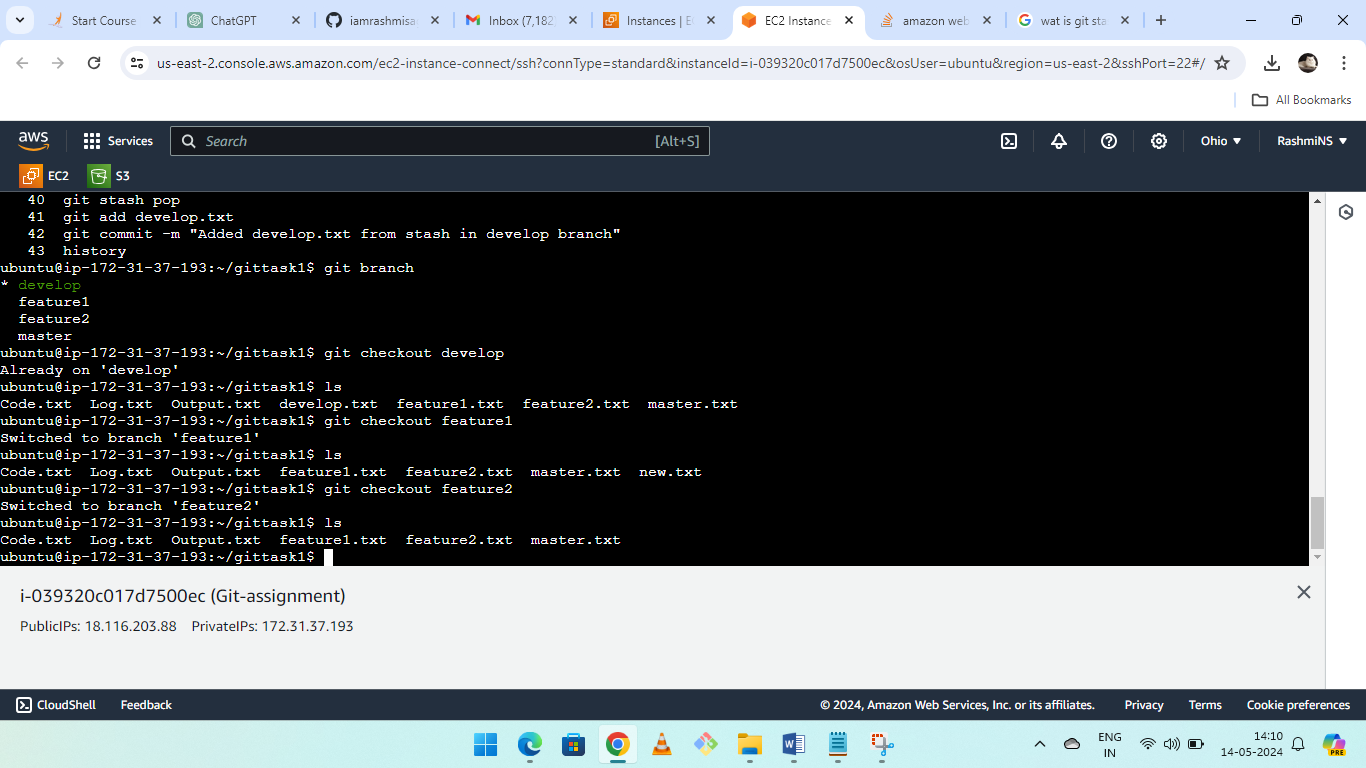
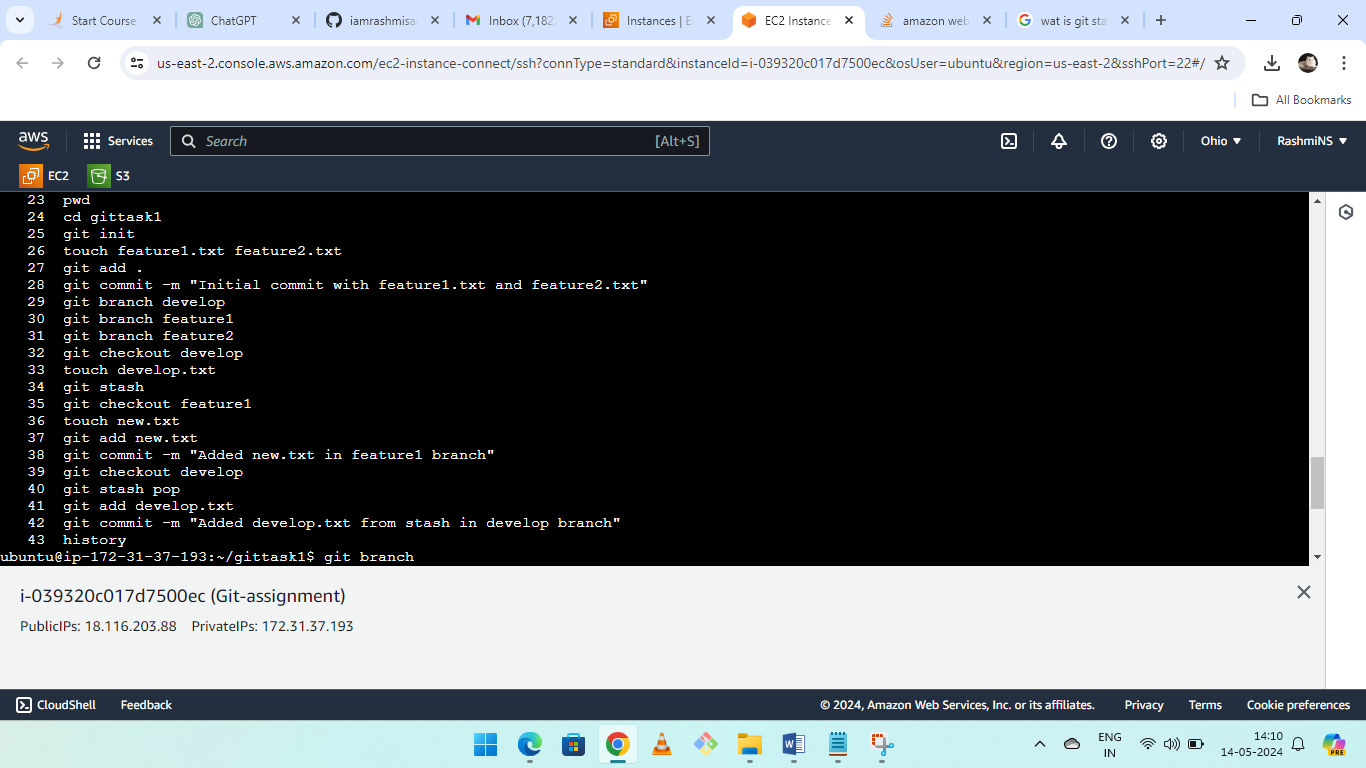
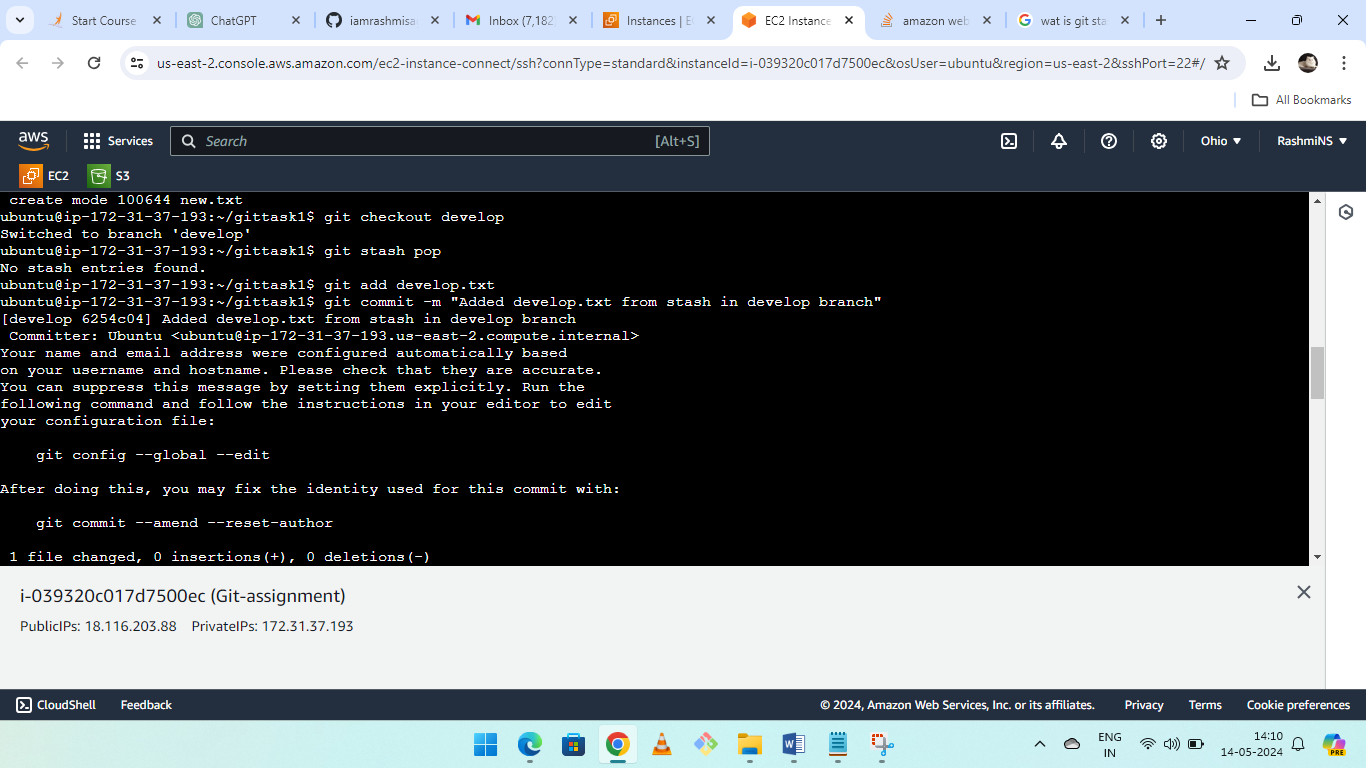
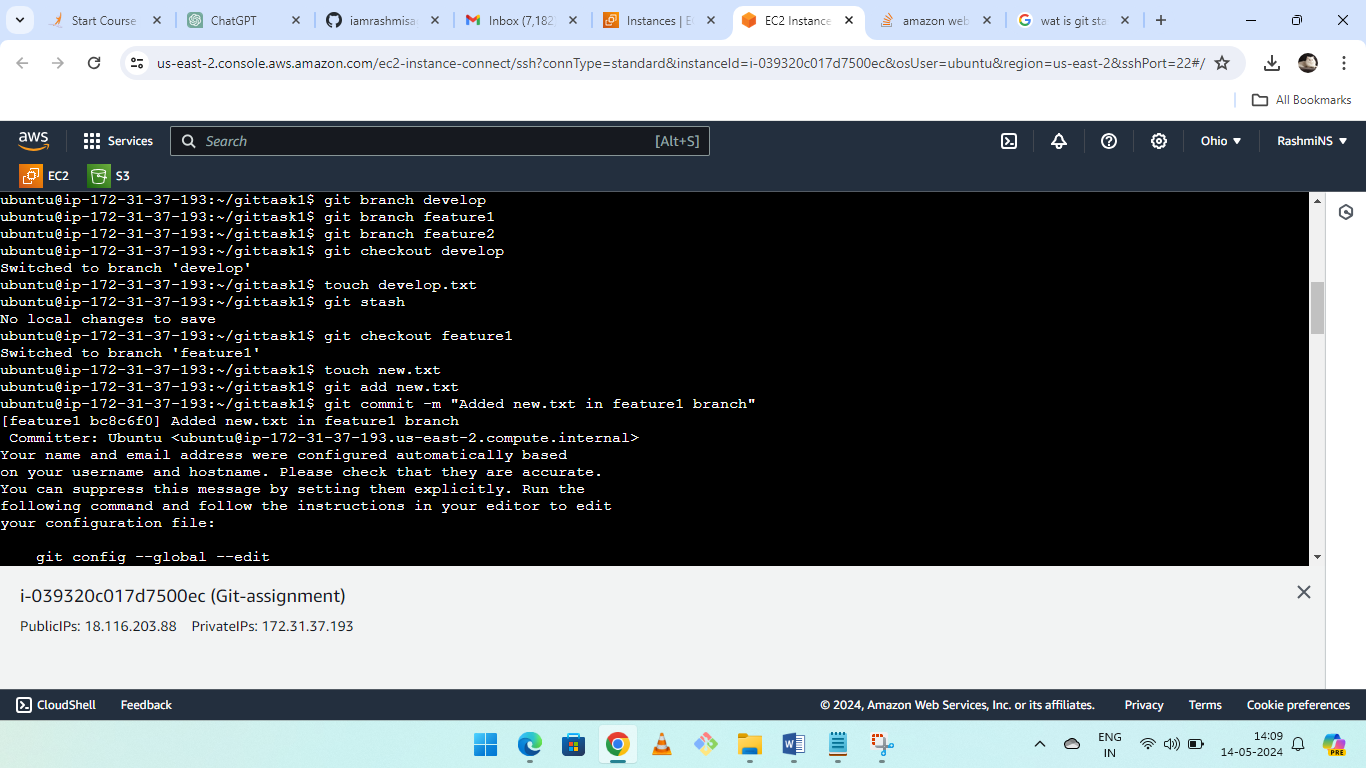
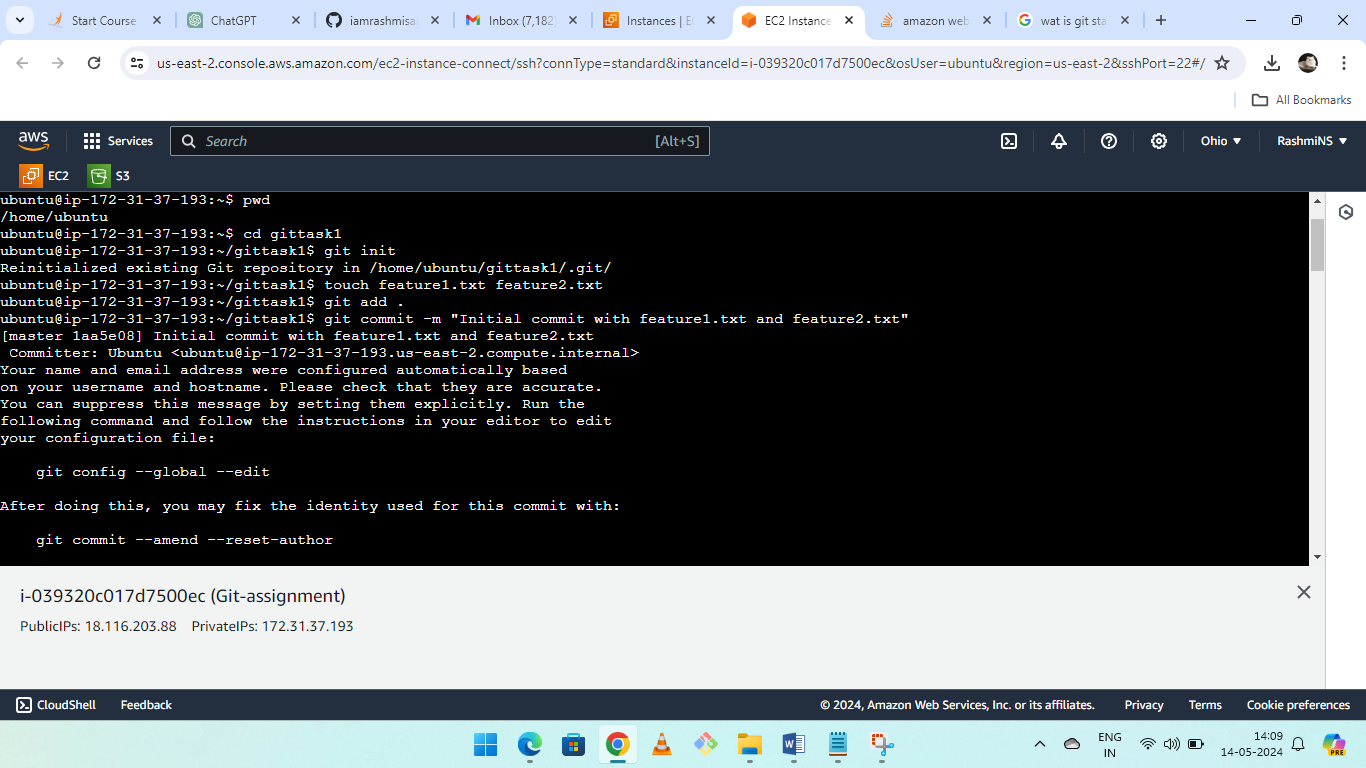
git commit -m "Added new.txt in feature1 branch"

git checkout develop

git stash pop

git add develop.txt

git commit -m "Added develop.txt from stash in develop branch"



Assignment3: ● Create a git working directory, with the following branches

○ Develop

○ F1

○ f2

● In the master branch, commit main.txt file

● Put develop.txt in develop branch, f1.txt and f2.txt in f1 and f2 respectively

● Push all these branches to github

● On local delete f2 branch

● Delete the same branch on github as well

**Assignment3 solution:**

mkdir gittask2

cd gittask2

git init

git checkout -b develop

git checkout -b F1

git checkout -b F2

touch main.txt

git add main.txt

git commit -m "Added main.txt file in master branch"

git checkout -b develop

touch develop.txt

git add develop.txt

git commit -m "Added develop.txt file in develop branch"

git checkout F1

touch f1.txt

git add f1.txt

git commit -m "Added f1.txt file in F1 branch"

git checkout F2

touch f2.txt

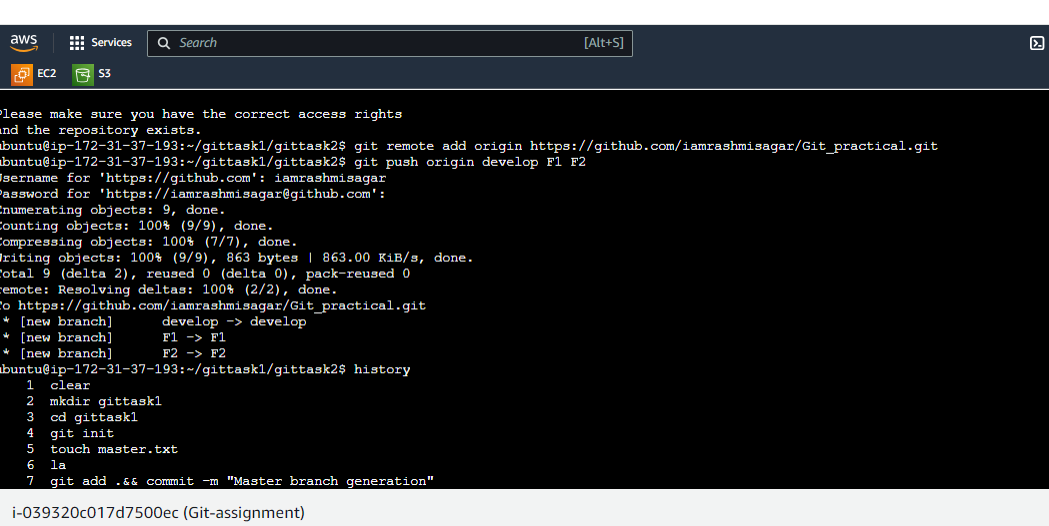
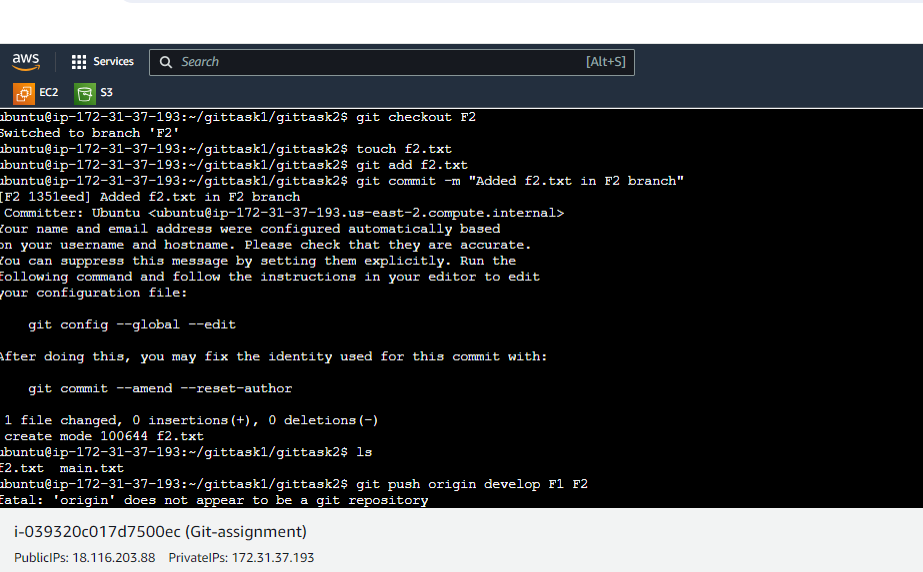
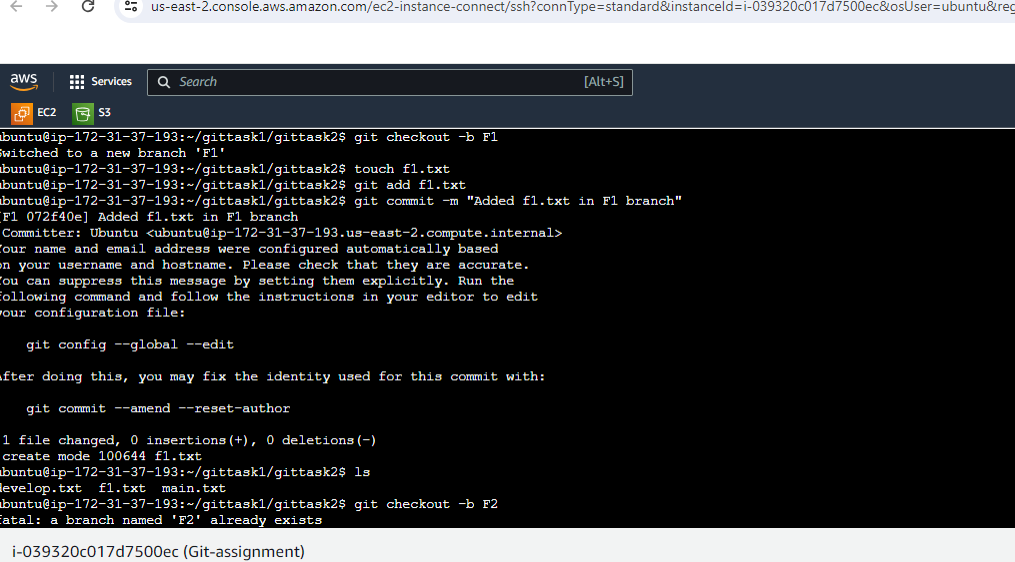
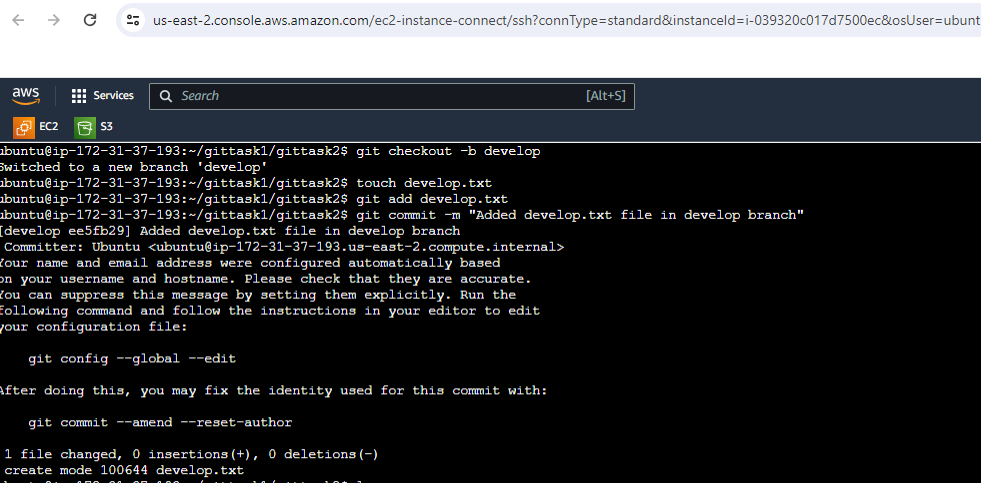
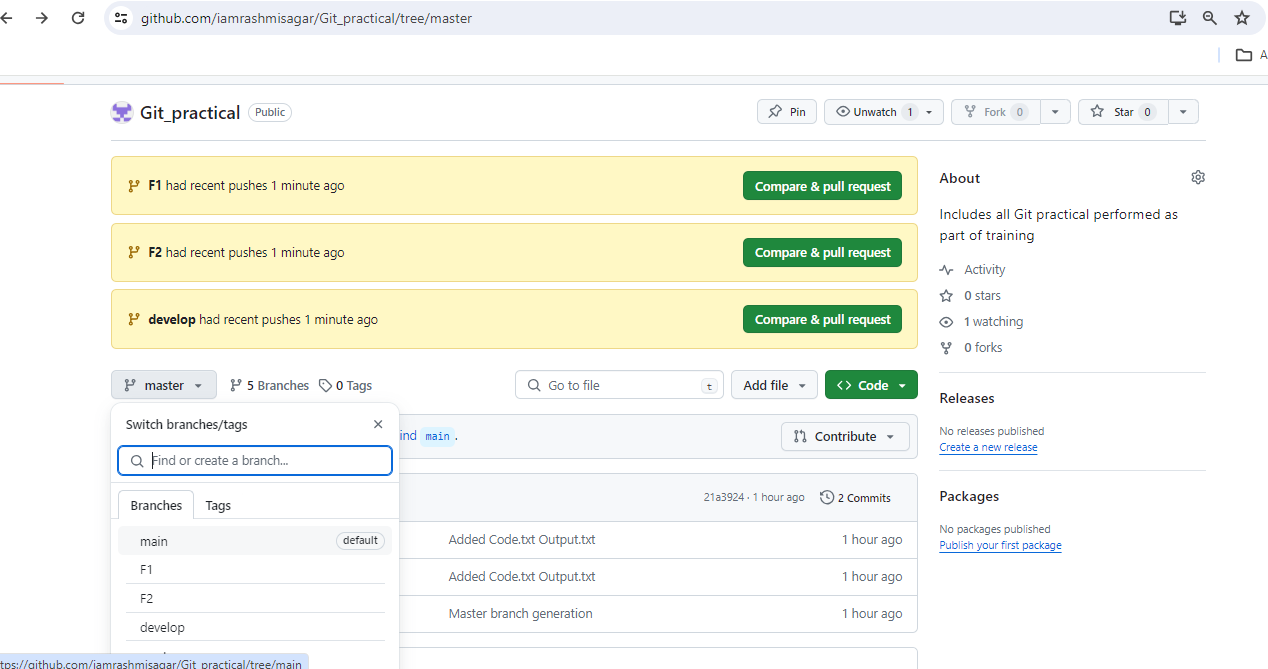
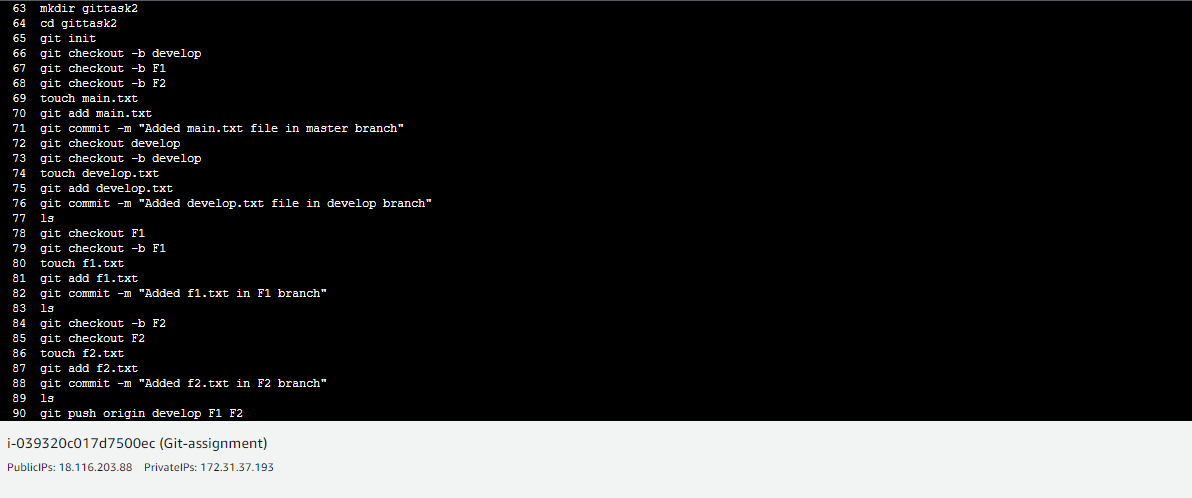
git add f2.txt

git commit -m "Added f2.txt file in F2 branch"

git push origin develop F1 F2

git branch -d F2

git push origin --delete F2



Assignment4: Put master.txt on master branch, stage and commit

● Create 3 branches: public1, public2 and private

● Put public1.txt on public 1 branch, stage and commit

● Merge public 1 on master branch

● Merge public 2 on master branch

● Edit master.txt on private branch, stage and commit

● Now update branch public 1 and public 2 with new master code in private

● Also update new master code on master

● Finally update all the code on the private branch

**Assignment4 solution:**

echo "This is master.txt" > master.txt

git add master.txt

git commit -m "Added master.txt on master branch"

git branch public1

git branch public2

git branch private

git checkout public1

echo "This is public1.txt" > public1.txt

git add public1.txt

git commit -m "Added public1.txt on public1 branch"

git checkout master

git merge public1

git merge public2

git checkout private

echo "Updated content in master.txt on private branch" > master.txt

git add master.txt

git commit -m "Updated master.txt on private branch"

git checkout public1

git merge master

git checkout public2

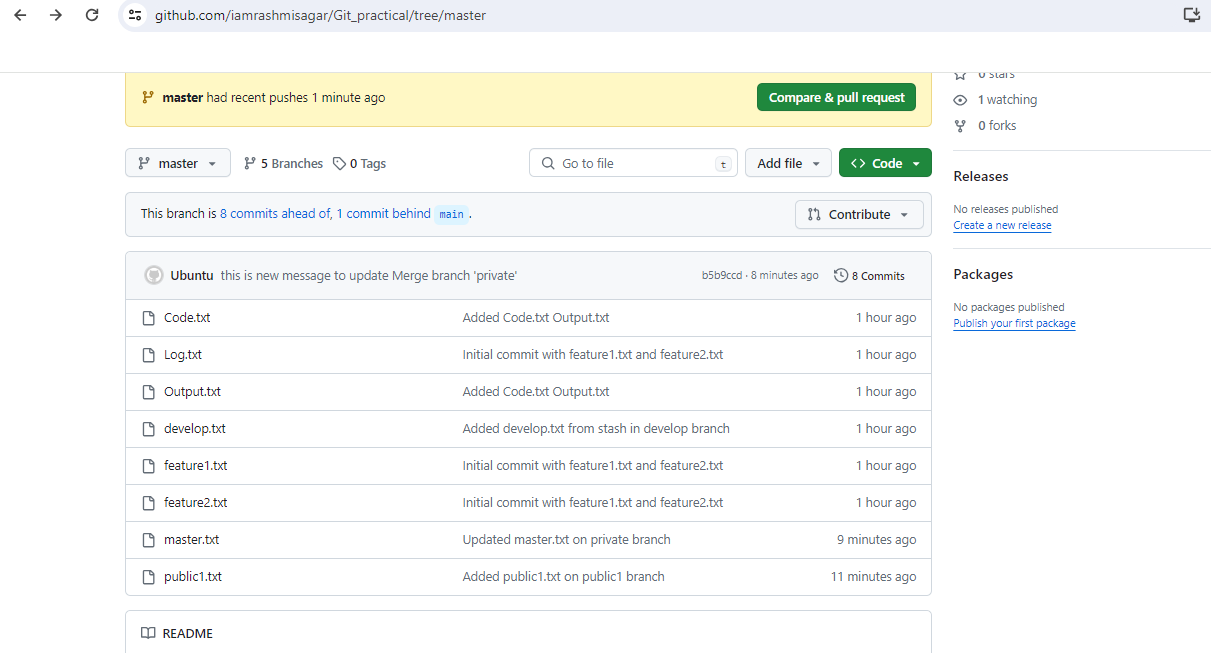
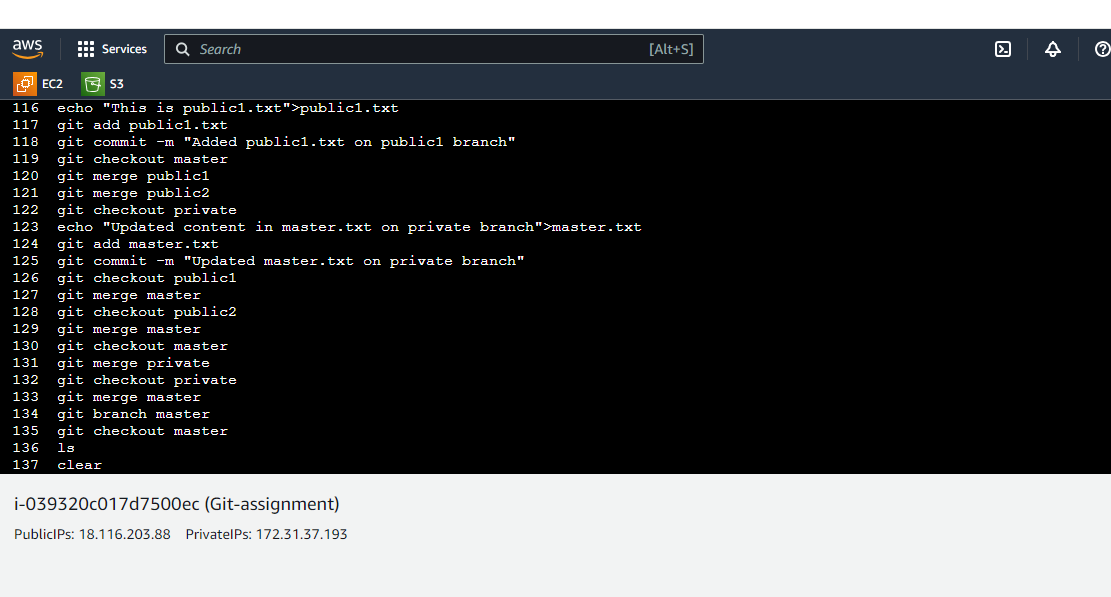
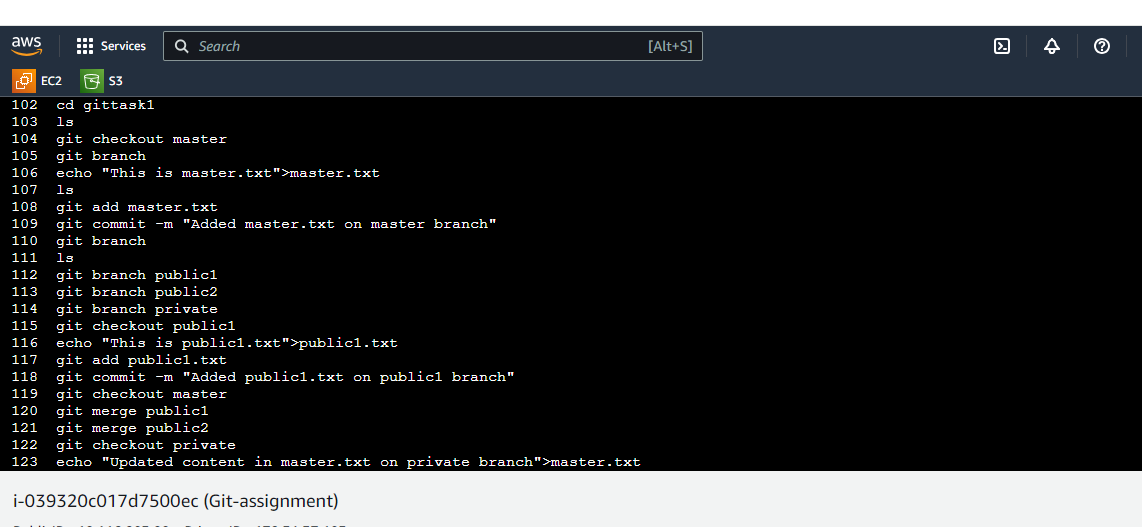
git merge master

git checkout master

git merge private

git checkout private

git merge master



Case study (Solution): Resolving merge conflict: Problem Statement:

You work for Zendrix Software & Co. You have been assigned the task of updating

the Master branch of their Git repository with all the features from the feature

branches.

Following is the GitHub account, https://github.com/devops-intellipaat/merge-conflict.git

Consider,

• Feature1 branch to be a public branch

• Feature2 branch to be a private branch

The company relies on a monolithic architecture, and for now all the code resides in one file

“main.c”.

The respective features have been added in the feature branches for main.c.

Meanwhile, a security patch was made to the master branch, and now feature1 and

feature2 branches are behind from master by 1 commit.

Following tasks have to be done:

1. Update Feature1 and Feature2 branch with the Security Patch

2. Apply changes of Feature1 and Feature2 branch on master

3. Finally push all the branches to GitHub

**Solution**:

Fork the repository **https://github.com/devops-intellipaat/merge-conflict.git** to my GitHub account

git clone <https://github.com/your-username/merge-conflict.git>

cd merge-conflict

git checkout Feature1

git pull origin master

git checkout Feature2

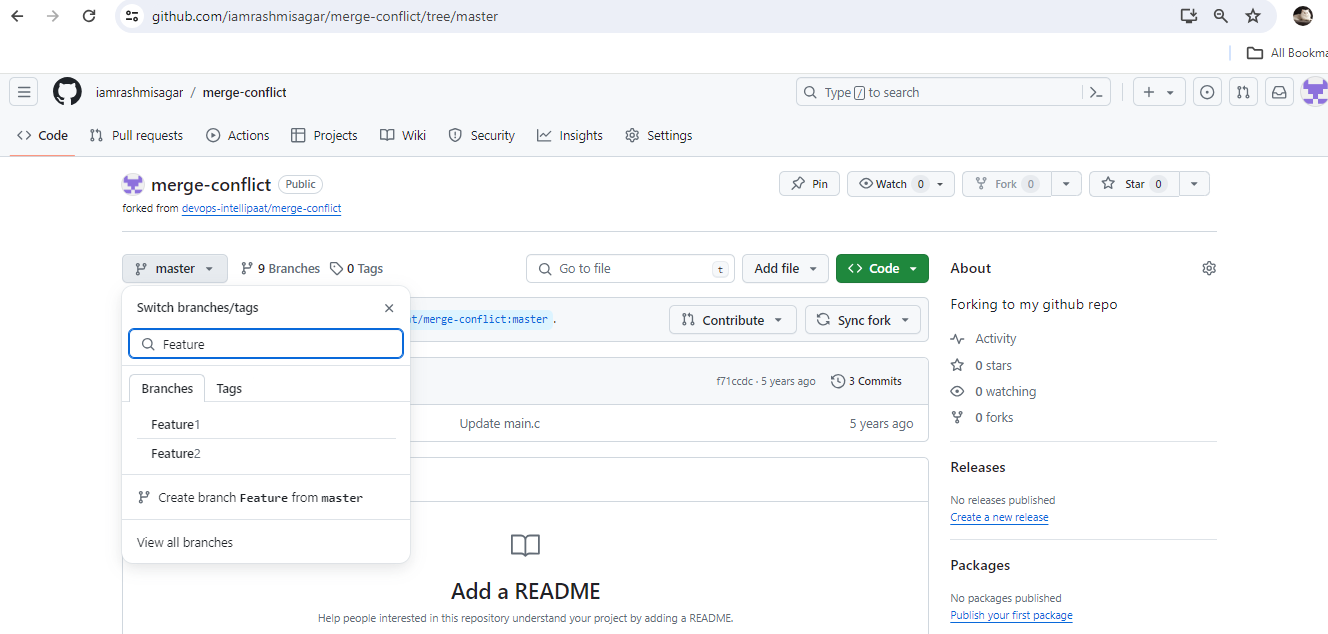
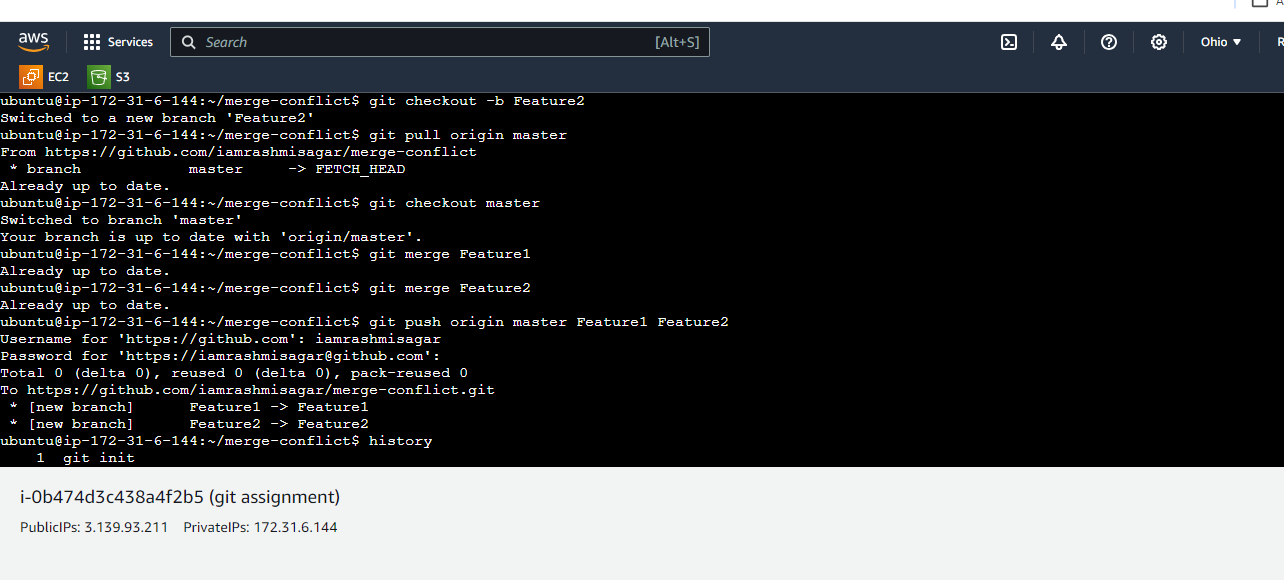
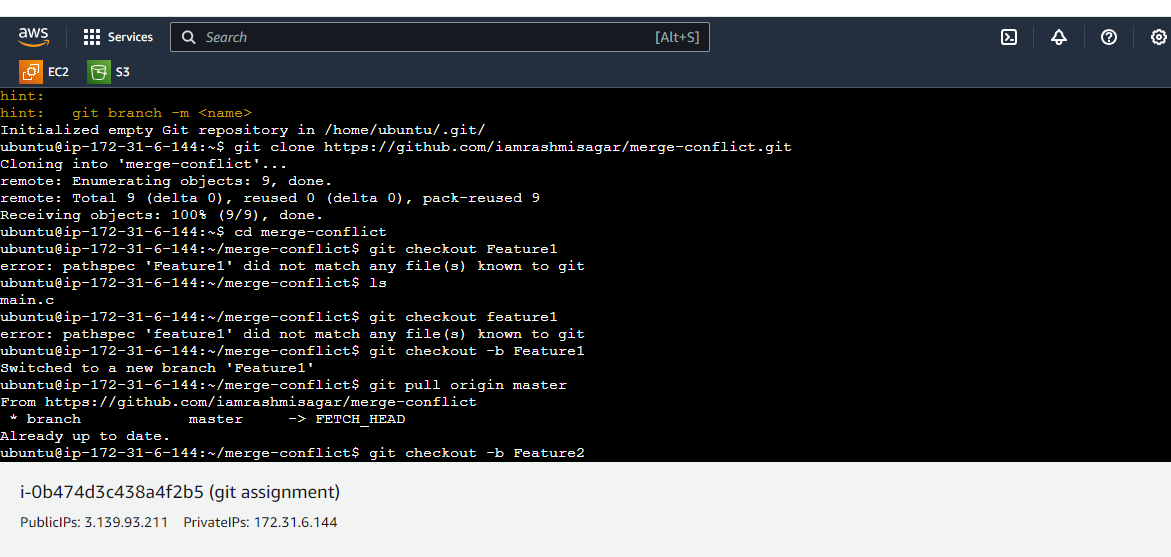
git pull origin master

git checkout master

git merge Feature1

git merge Feature2

git push origin master Feature1 Feature2



**Case Study2- Git Workflow Architecture (Zendriix Software):**

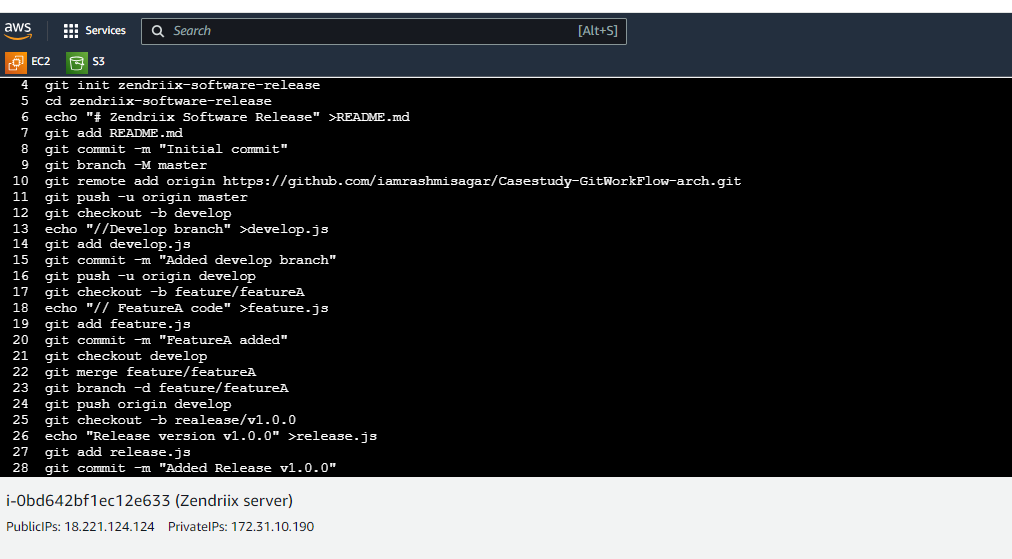
Problem Statement: You work as a Devops Architect in Zendriix Softwares. The company has been struggling to manage their product releases. The releases should happen on 25th of every month. Suggest a Git Workflow Architecture for this requirement. Simulate this workflow, by creating a pseudo code files and branches, and upload the same to your GitHub Account. As a part of solution, share the link to your GitHub repository.

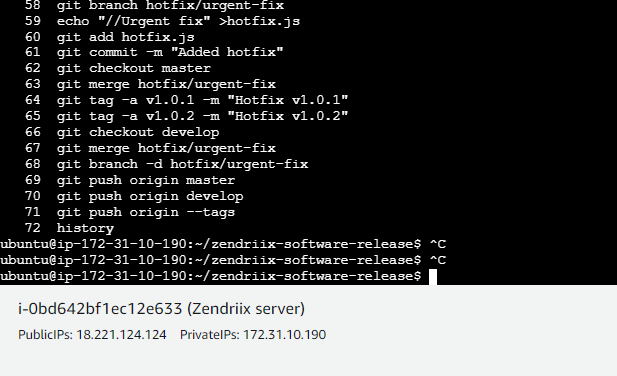
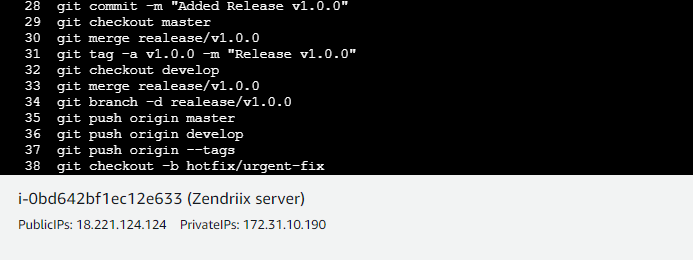
**Solution:**

We have Git repo created for this simulation: Casestudy-GitWorkFlow-arch

Workflow architecture involves steps as follows:

1. Creating master branch
2. Creating Develop branch
3. Creating supporting branches like:
   1. Feature branch
   2. Release branch
   3. Hotfix branch

Commands used: 



You can check the simulated workflow on this GitHub repository: <https://github.com/iamrashmisagar/Casestudy-GitWorkFlow-arch.git>