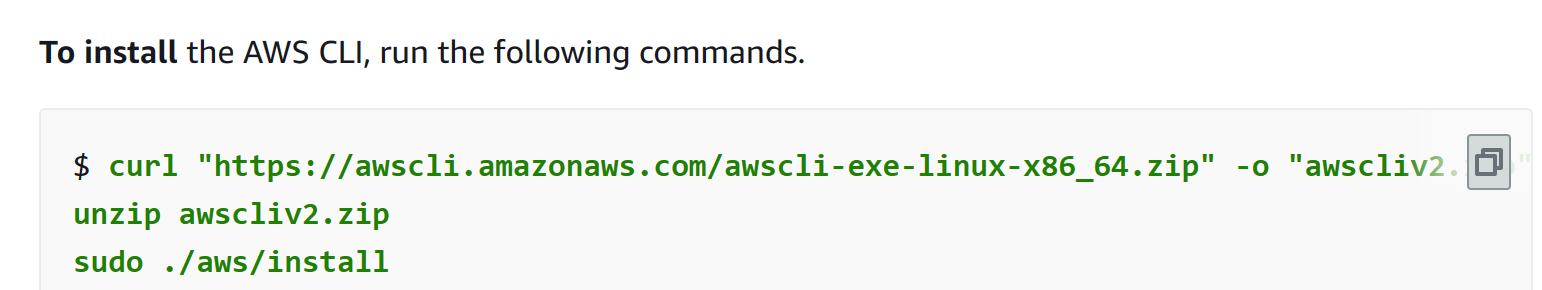
**Project 1 – Creating EC2 Instance using terraform**

**Step – 1:** Create an Ubuntu EC2 instance. Install GIT, AWS CLI & Terraform.

Commands: **git --version (**Installed by default**).**

If not then **sudo apt install git -y 🡪 sudo apt install unzip**

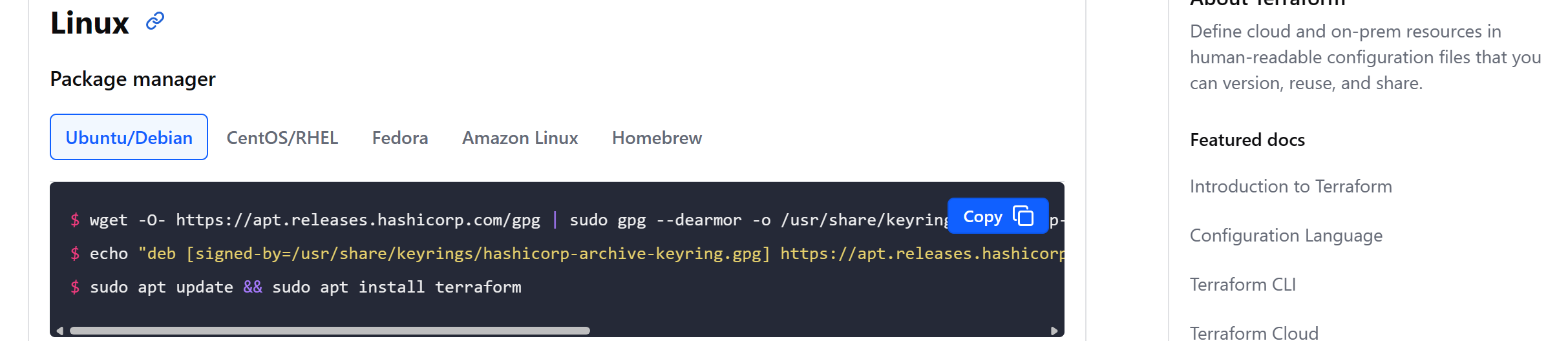
For AWS CLI:[**https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-instructions**](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-instructions)

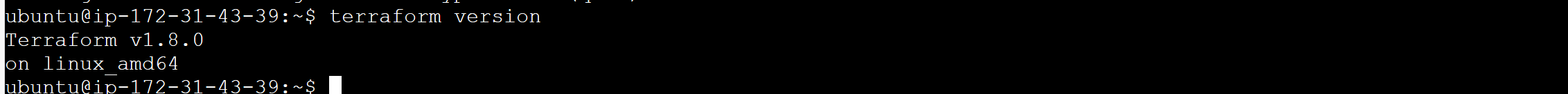
****Add sudo for second command in above picture.

Can give **aws version** to check the installation success or not.

**Installing Terraform:**

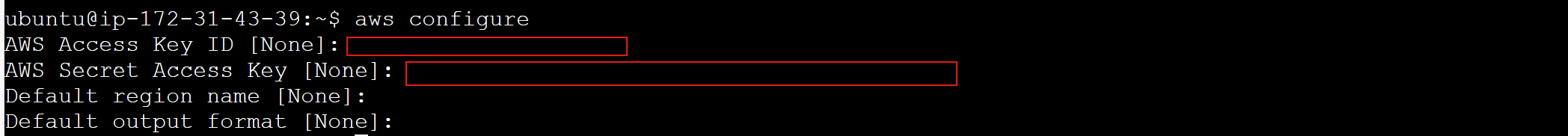
Create an Ubuntu EC2 Instance – Follow the commands in here: <https://developer.hashicorp.com/terraform/install#linux>



Copy those commands from there, paste and execute in instance. 

**Step – 2:** Configuring AWS User in AWS CLI.

**Configuring AWS User in AWS CLI:** Go to Aws website and on top right corner click on your account name and then security credentials. Now create Access key and download the csv file. After this go to the command line and give “**aws configure**”. Enter access key keys and go with default values for next two fields. Now aws cli is setup and configured. Thought this CLI we should be able to create, use and manage aws services.



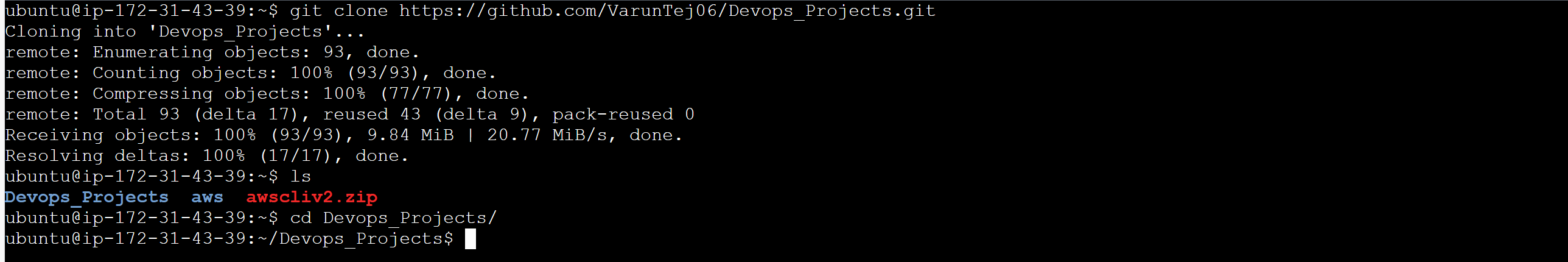
**Step – 3:** Cloning GitHub Repo to which we want to push our Terraform Project.

Reference GitHub URL of Project: <https://github.com/iam-veeramalla/write_your_first_terraform_project/tree/main>

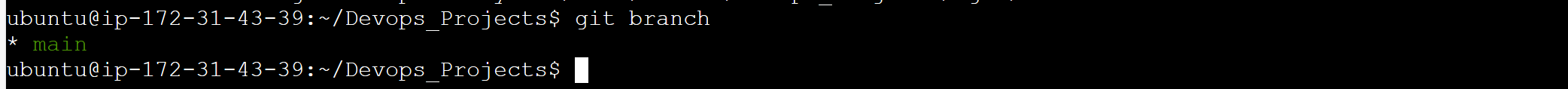
In this repo, we need “main.tf” in local\_state folder and all the files in remote-state folder. These two folders are two different projects.

We are not cloning this repo as we wanted to do it from scratch and upload to our repo.

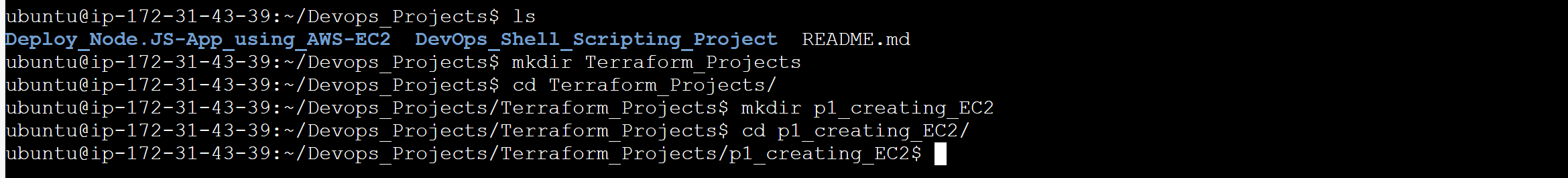
**Repo to be cloned:** <https://github.com/VarunTej06/Devops_Projects>

****Commands: **git clone repo\_url 🡪 ls 🡪 cd Devops\_Projects/**

**Step – 4:** Now initialize git repo in this current directory.

Command:  **“git init .”** **** 

**Step – 5:** Create directories for Terraform Project

Give “ls”: we have two projects as of now in that repo.

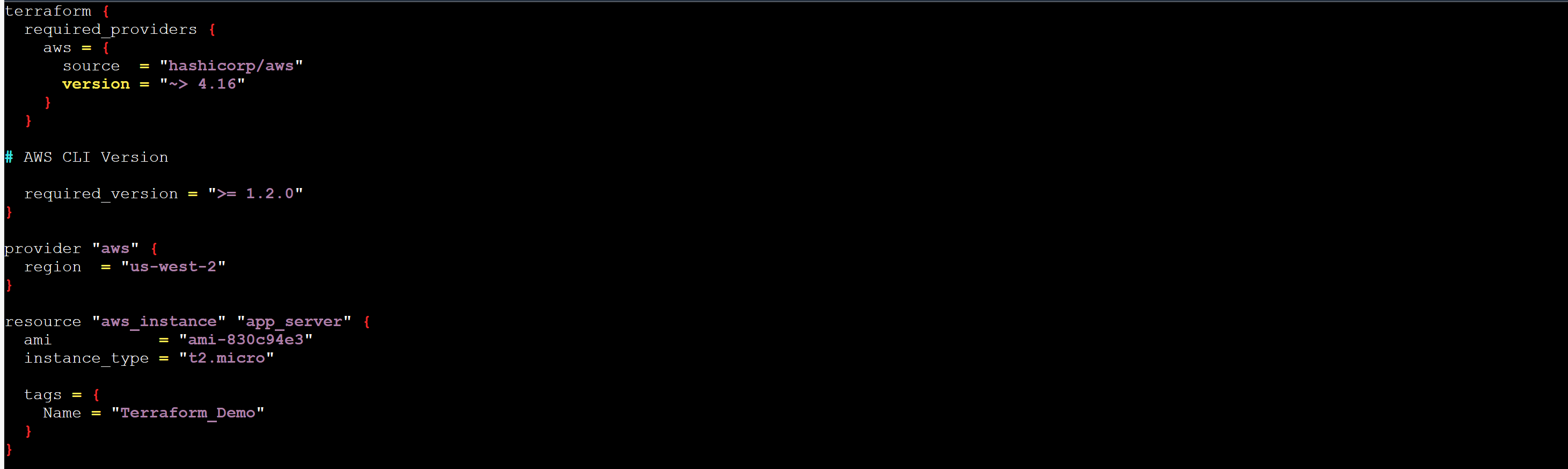
Create a new directory for storing Terraform project.

Commands: **mkdir Terraform\_Projects 🡪 cd Terraform\_Projects/ 🡪 mkdir p1\_creating\_EC2 🡪 cd p1\_creating\_EC2/**

**Step – 6:** Writing terraform file “main.tf “to create EC2 instance.

Commands: **vim main.tf**

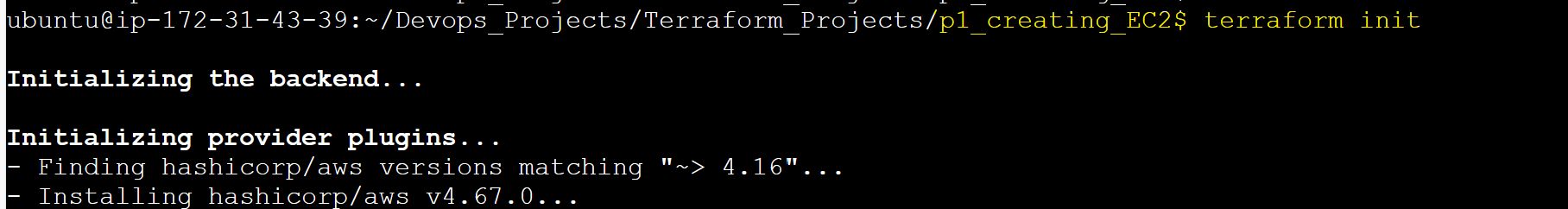
Copy the code from below path to the main.tf file. 



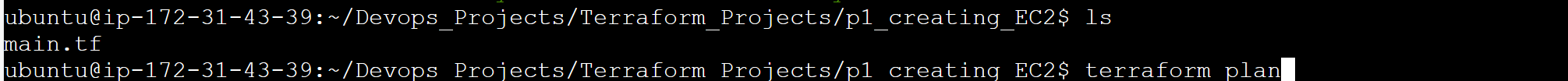
Now, we have terraform file to create EC2 instance. We have to execute this now to create EC2.

**Step – 7:** Initialize Terraform now in the Project folder. Here our project folder is “p1\_creating\_Ec2”.

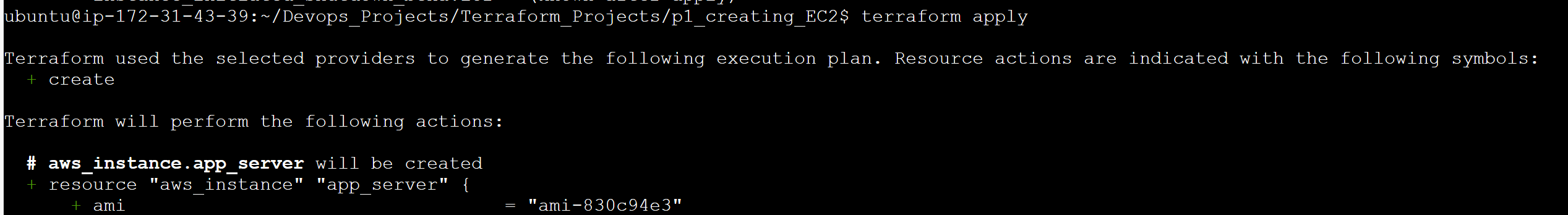
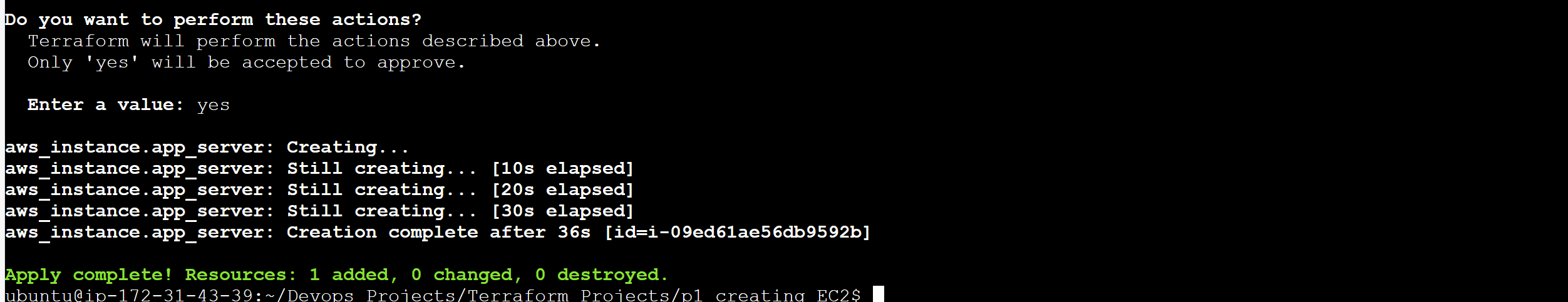
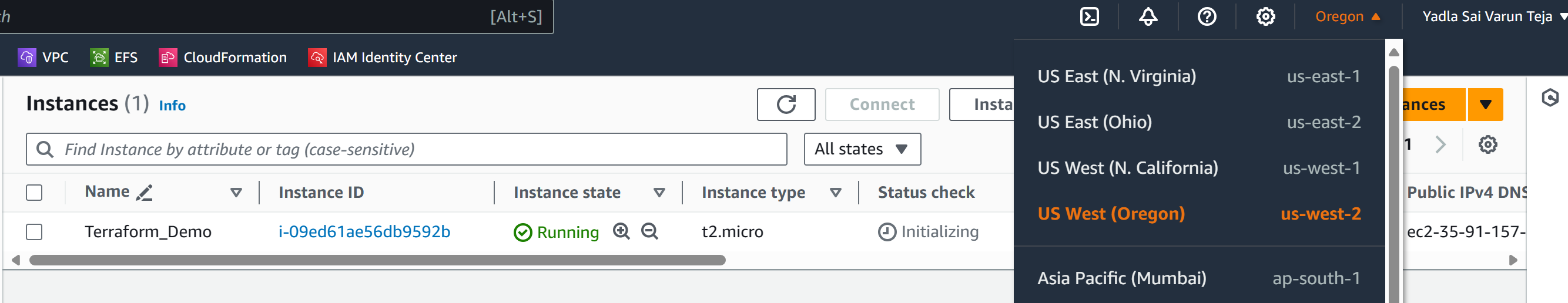
Command: **terraform init**



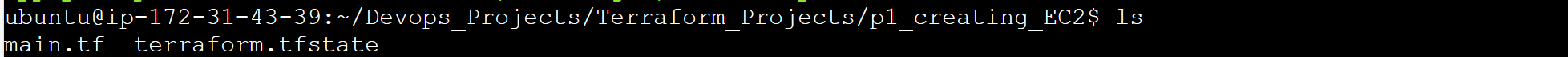
**Step – 8:** Do a dry run to check what will be changes we are doing with the terraform file.

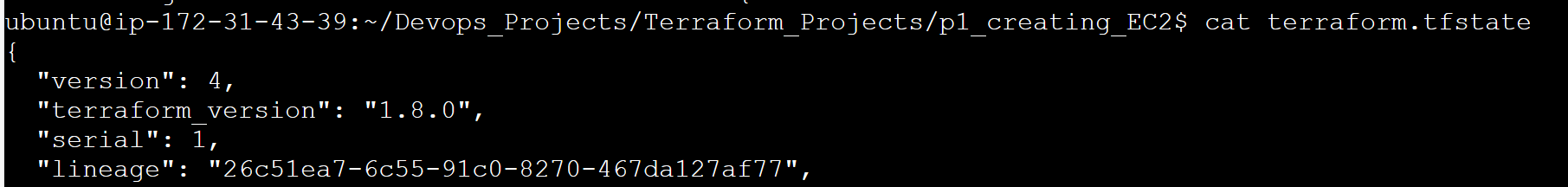
Command: **terraform plan** **** 

**Step – 9:** Creating EC2 instance

Command: **terraform apply**  Check the AWS EC2 console whether EC2 is created or not. EC2 instance created in us-west-2 region as per the terraform code.

**Step – 10:** Knowing about terraform state file

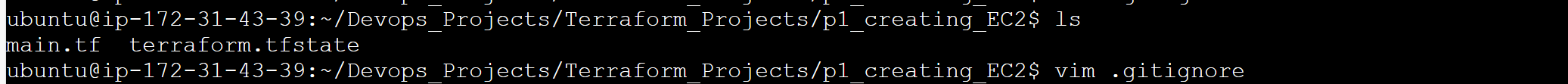
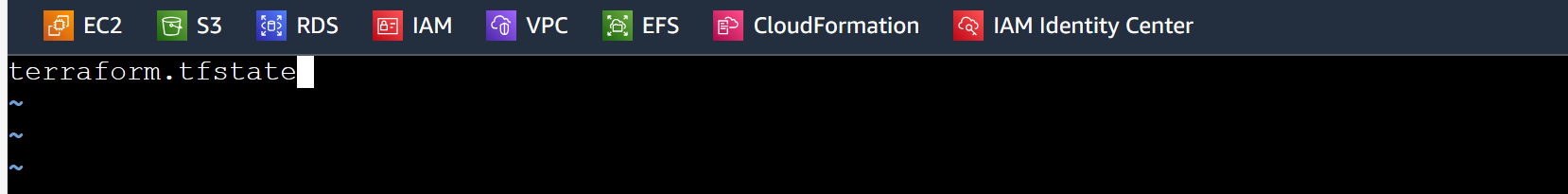
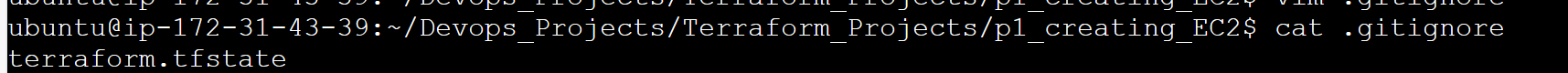
Command: **ls** 🡪Once we give apply command, it will create **terraform state file** which will have the data of the infra it created with terraform config file. In this it will track the infrastructure. Terraform will track all the changes here. It will have all the sensitive and non-sensitive information.

Now, it will have the data of the EC2 instance we created. **This file should not be moved to GitHub.**

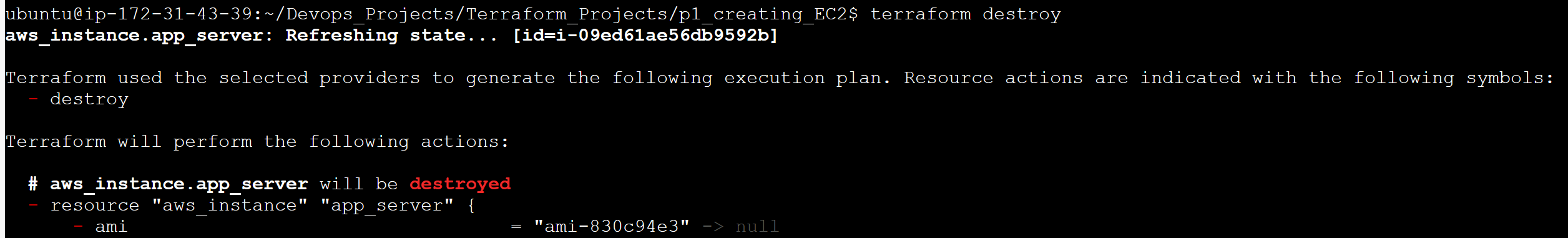
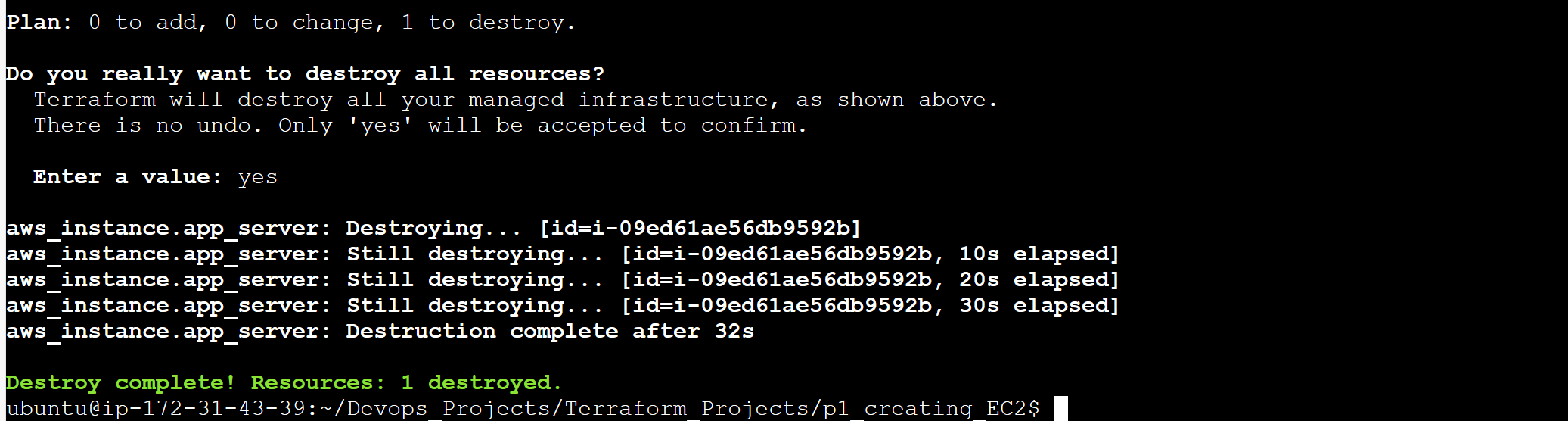
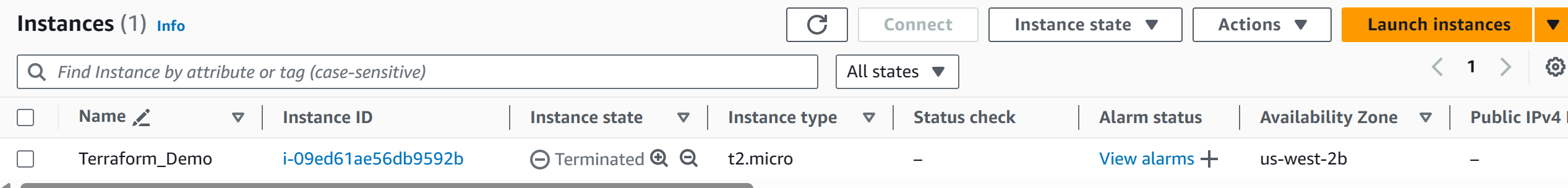
**Step – 11:** Ignoring **terraform.tfstate**

Since we should not push **terraform.tfstate** file, create gitignore file in the same directory where you want to ignore the files.

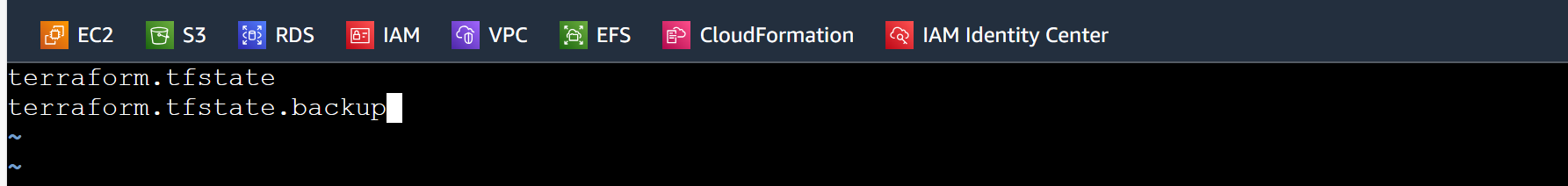
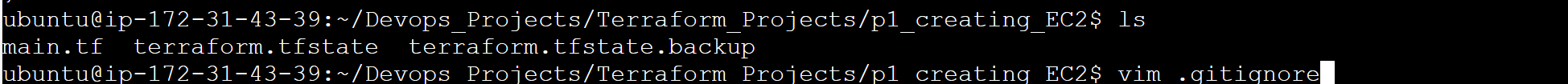
Current Directory: p1\_creating\_EC2 (Same directory from begining).

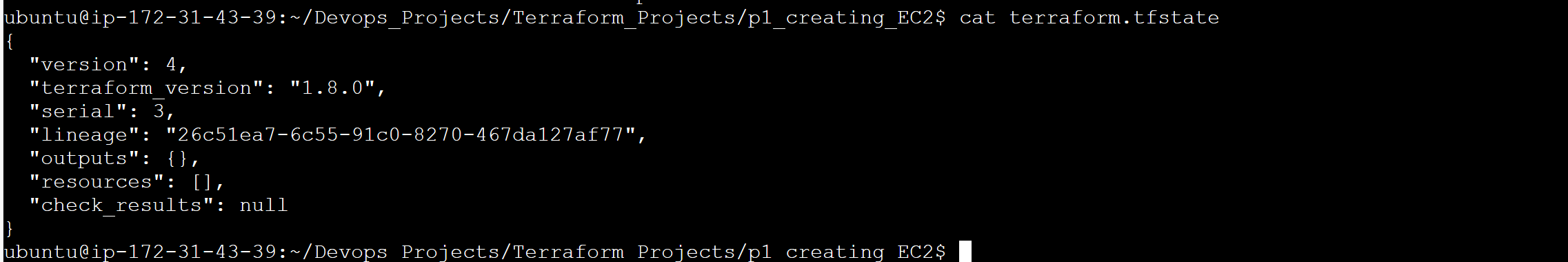
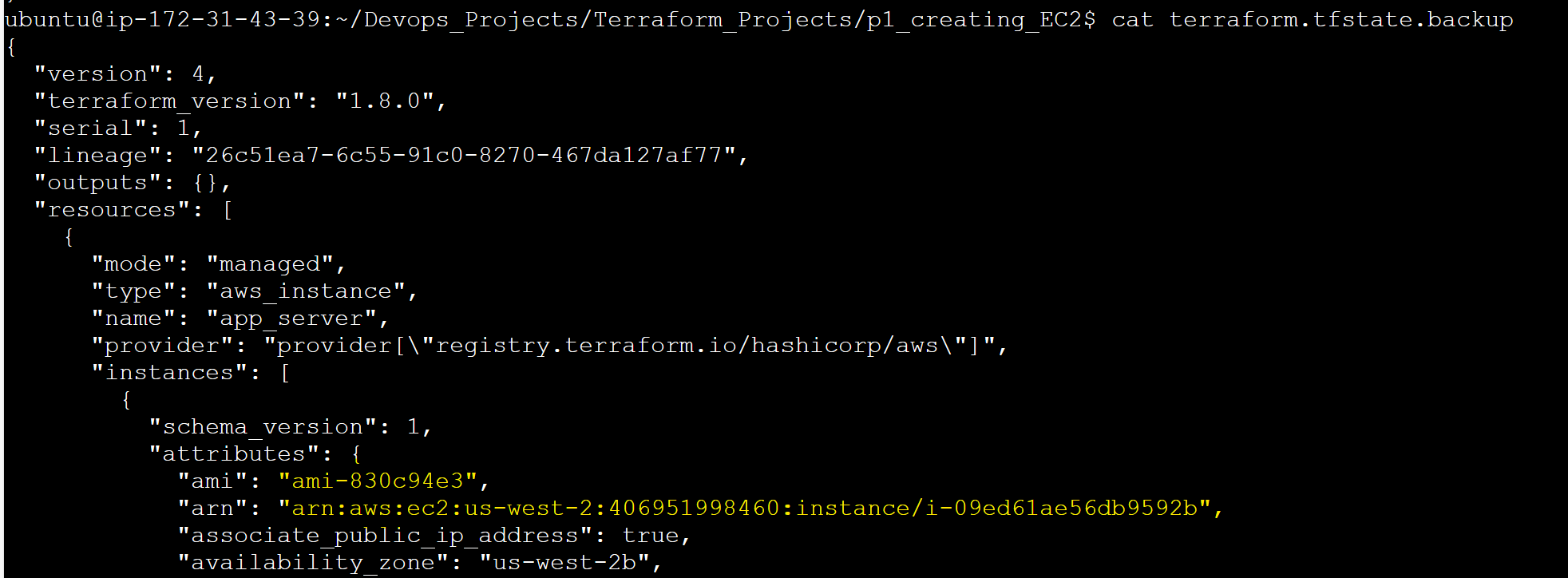
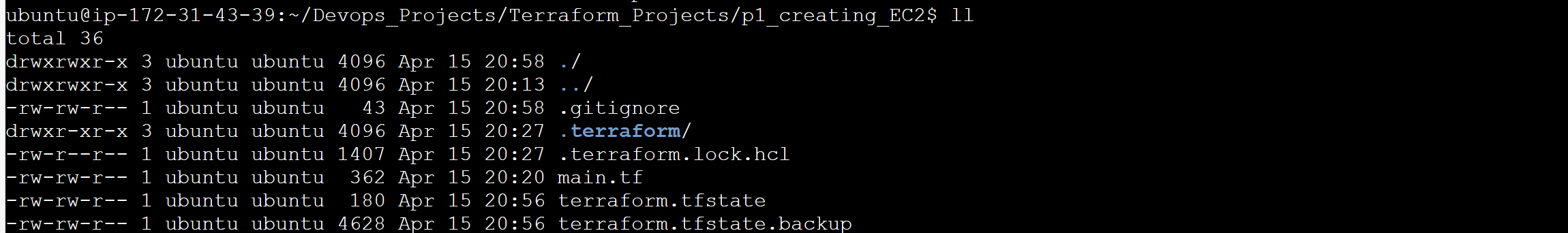
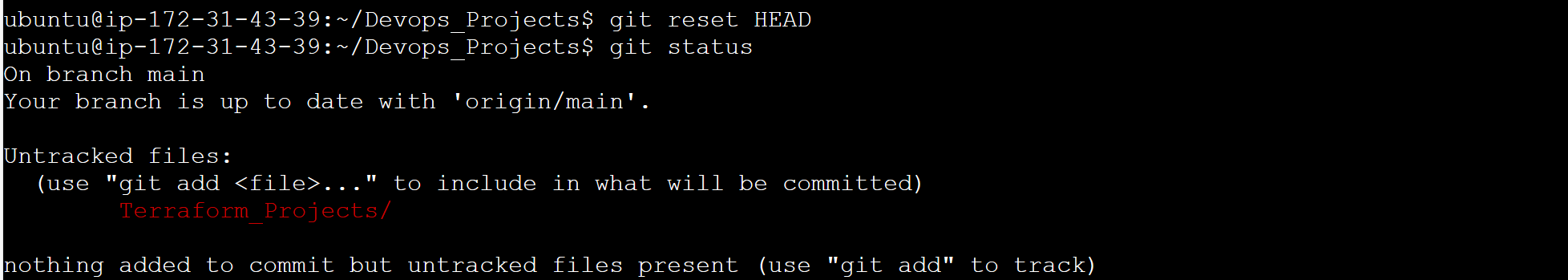
Command: **vim .gitignore** **** 

**Step – 12:** Destroy the Ec2 instance created.

Command: **terraform destory** 🡪 This command will destroy whatever that is created through apply command.  Check in EC2 console it is terminated or not. 

**Step – 13:** Updating the “.gitignore” file

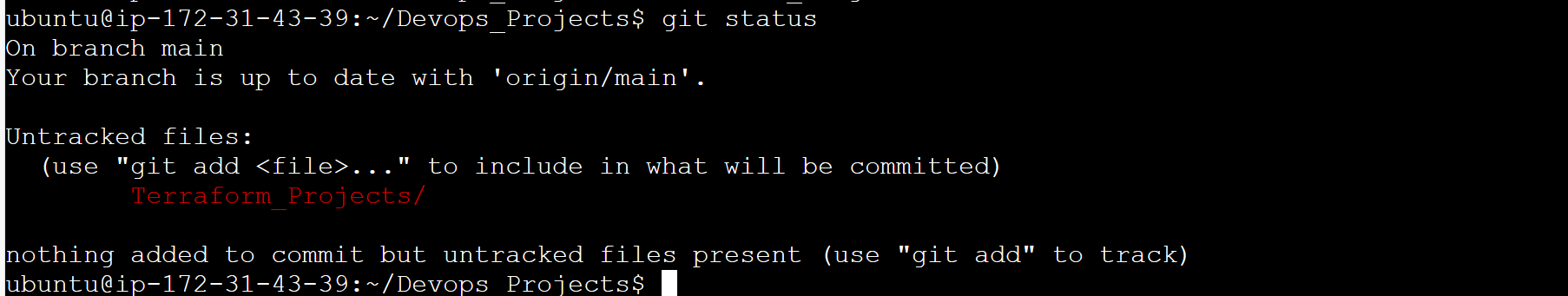
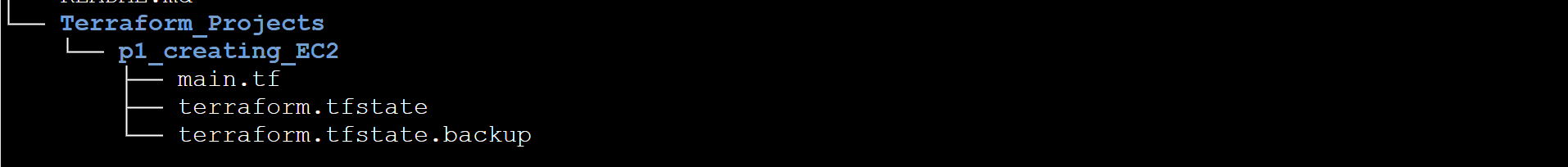
After **terraform destroy** it created another file called **terraform.tfstate.backup** it will have the data that is present in **terraform.tfstate** file before and **terraform.tfstate** file will have updated infra config after destory command which means nothing will be there. 

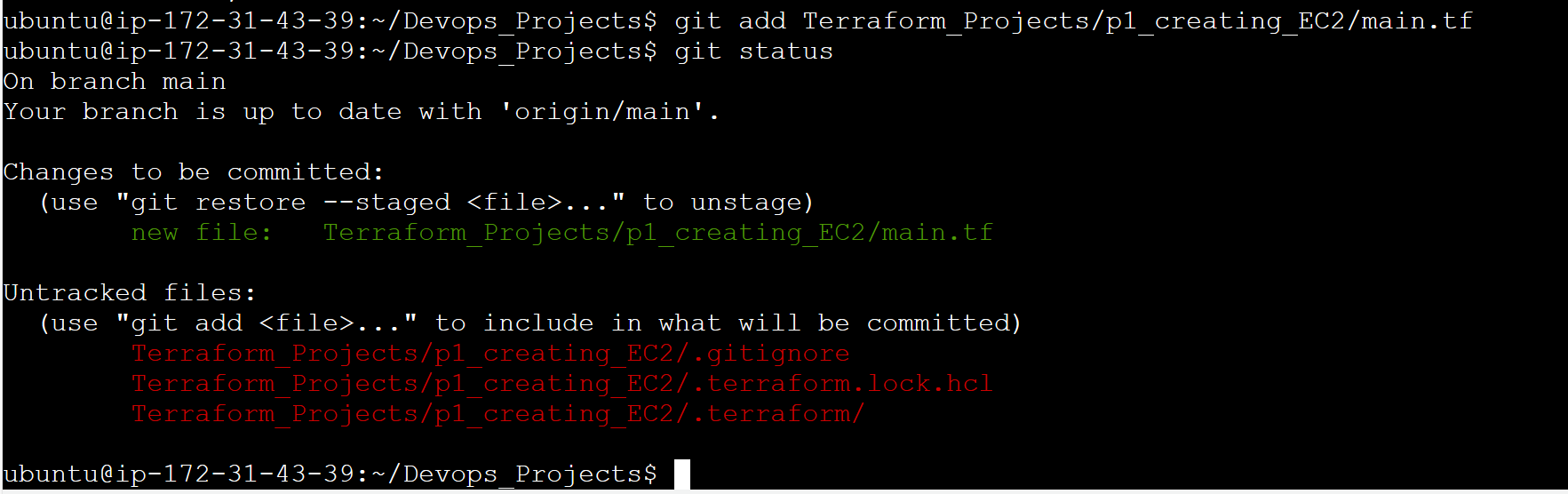
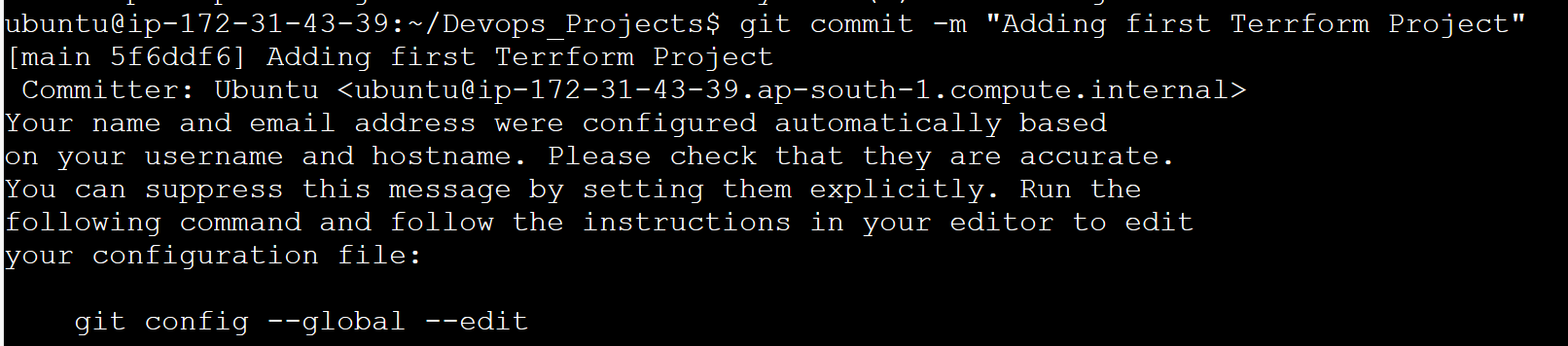
Updated **terraform.tfstate** file: Newly created, **terraform.tfstate.backup** file after **terraform destroy: **We only need “main.tf” file into our GitHub repo. Because we can us ethe same main.tf file and try this project again. We can also push “.gitignore” file as well. If we clone this project next time, this .gitignore file will also be cloned along with main.tf. No issues will cause due to that. If you try next time and tries to push whole project folder then it will automatically ignore what is there in “.gitignore”. There are many other files as well other than main.tf and gitignore file. Those also needs to be ignored. Otherwise if you give **git add \*** it will add all the files to staging area along with hidden files.To unstage the tracked files, give “**git reset HEAD .”**

**Step – 14:** Commit the project to Git Repo.

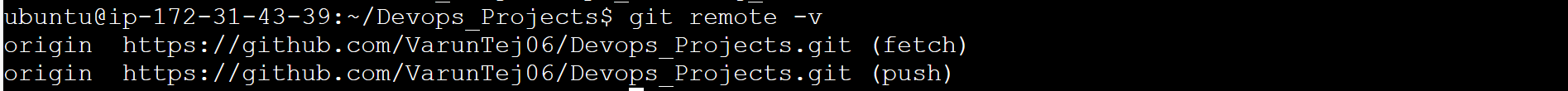
First go back to the directory where we initiatised the git. Directory: Devops\_Projects

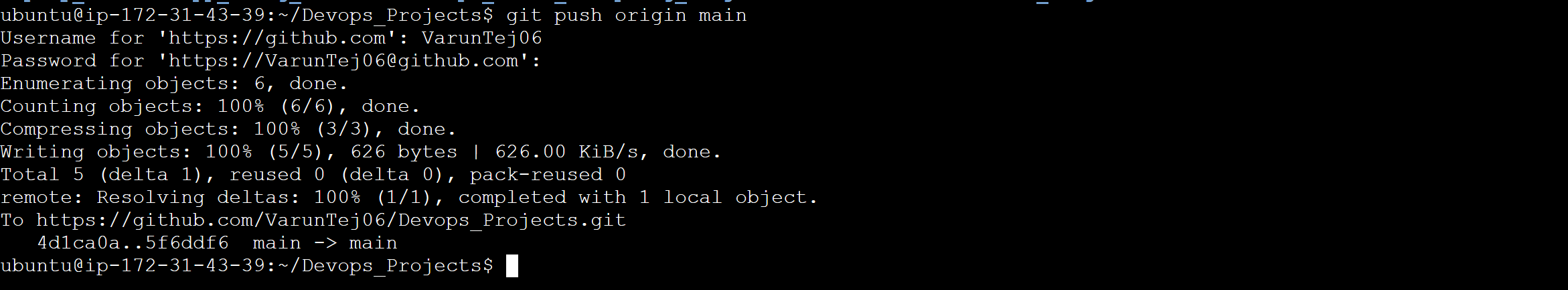
Command: **cd .. 🡪 cd .. 🡪 git status**

Now give tree command. If tree not present install with **sudo apt install tree**Now, give this git command to add main.tf to staging area.

Command: **git add Terraform\_Projects/p1\_creating\_EC2/main.tf** ****Now, commit the changes to git repo with **git commit -m “Adding first Terraform Project”** ****

**Step – 15:** Push the Project to GitHub repo.

Command: **git remote -v** 🡪 to know the name of remote repos which are already in connection to the git repo. 

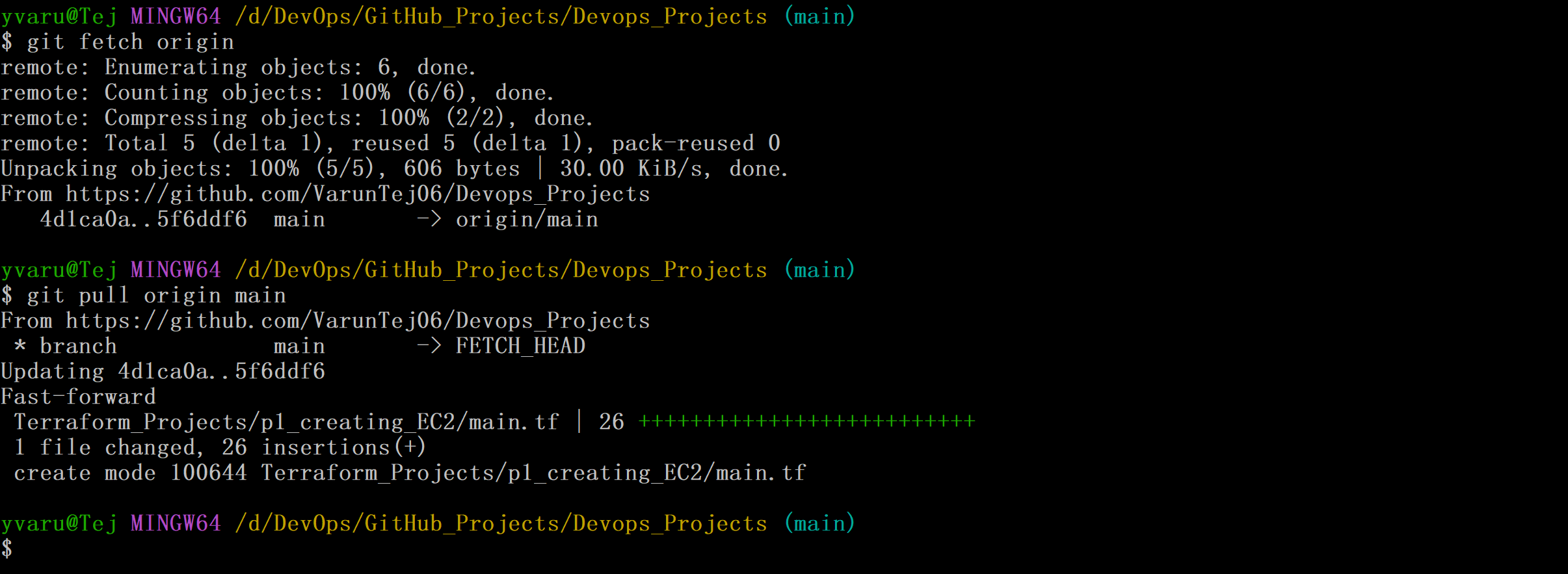
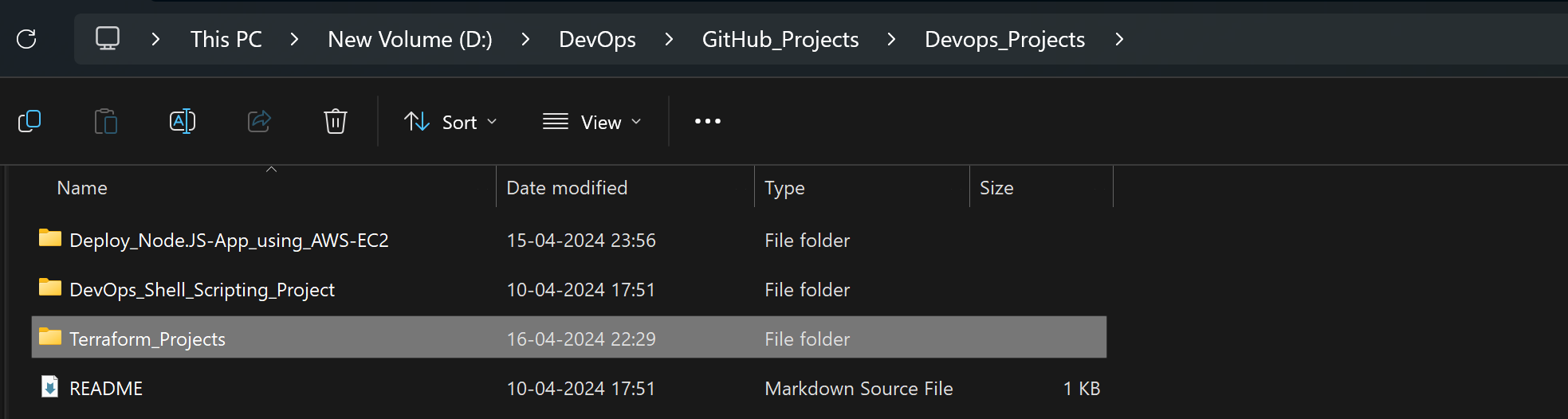
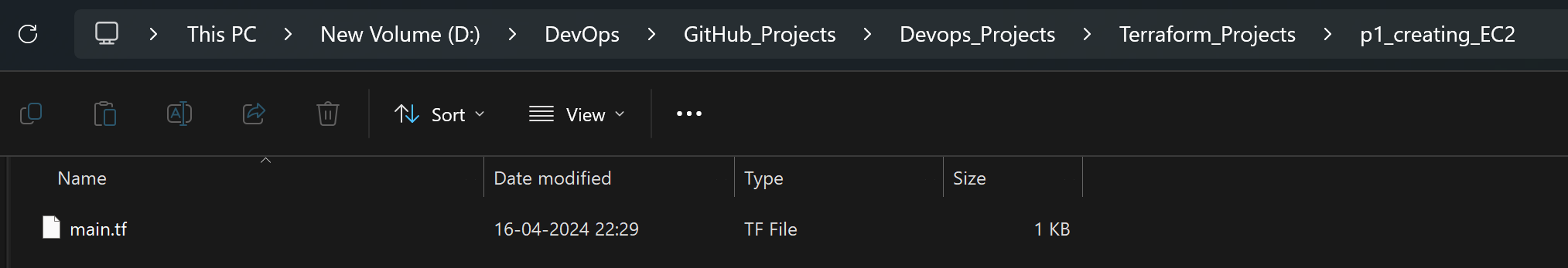
Command: **git push origin main** ****Since we added MFA to our GitHub account when we push it is asking for Username and password. If you enter the passowrd of GitHub account it will fail because of MFA. You have to give access token. Then it will push the changes to GitHub repo successfully.

**Additional Steps:**

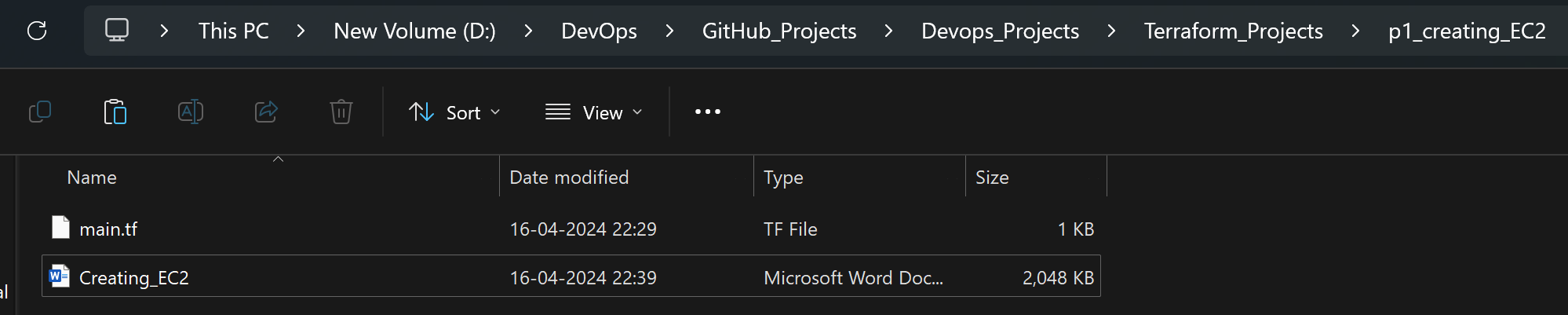
**Step – 16:** Now sync the GitHub Devops\_Projects repo with local PC Repo.

Go to directory where you initiated git in your local for GitHub repo as a Project and open Git Bash or CMD Prompt.

Directory: GitHub\_Projects (This will be project repo where we initialized git repo to continuously update local and GitHub repo with my projects).

Commands: **git fetch origin 🡪 git pull origin main** ****  Local Folders are updated now and in sync with GitHub repo.

**Step – 17:** Adding Step by Step guide for this terraform Project from local to GitHub repo.

Copy the document to the target folder. I copied and pasted the doc in p1\_creating\_EC2 folder. Now go to the same Git Bash in step -16 and give **git add \* 🡪 git status (**only one file**) 🡪 git commit -m “Adding Project guide” 🡪 git push origin main** to push this file to GitHub repo.