DEVSECOPS : Integrating Security in DevOps Practices Lab

Lab Report

Autumn 2025

Lab experiment <exp_no - 1>

Name: Pulkit

Sap_ID: 500125835

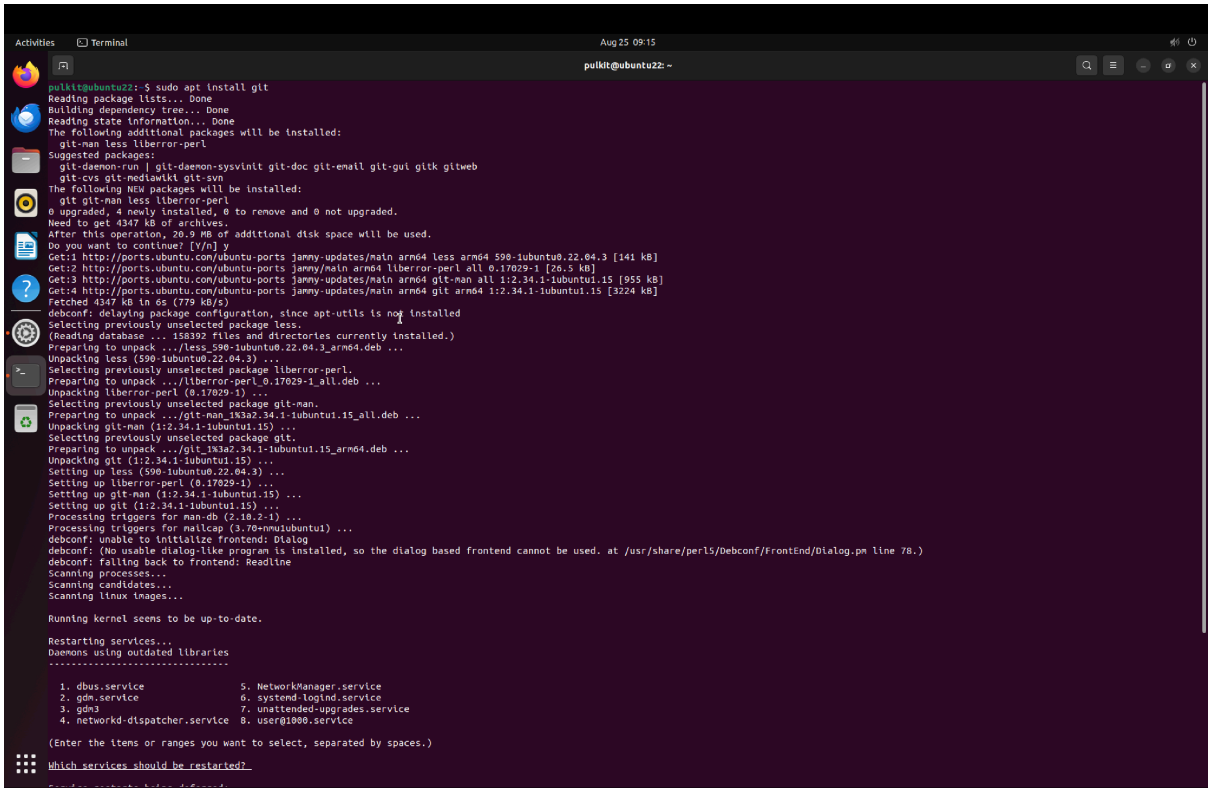
Date of experiment: 11/08/25

School of Computer Science,

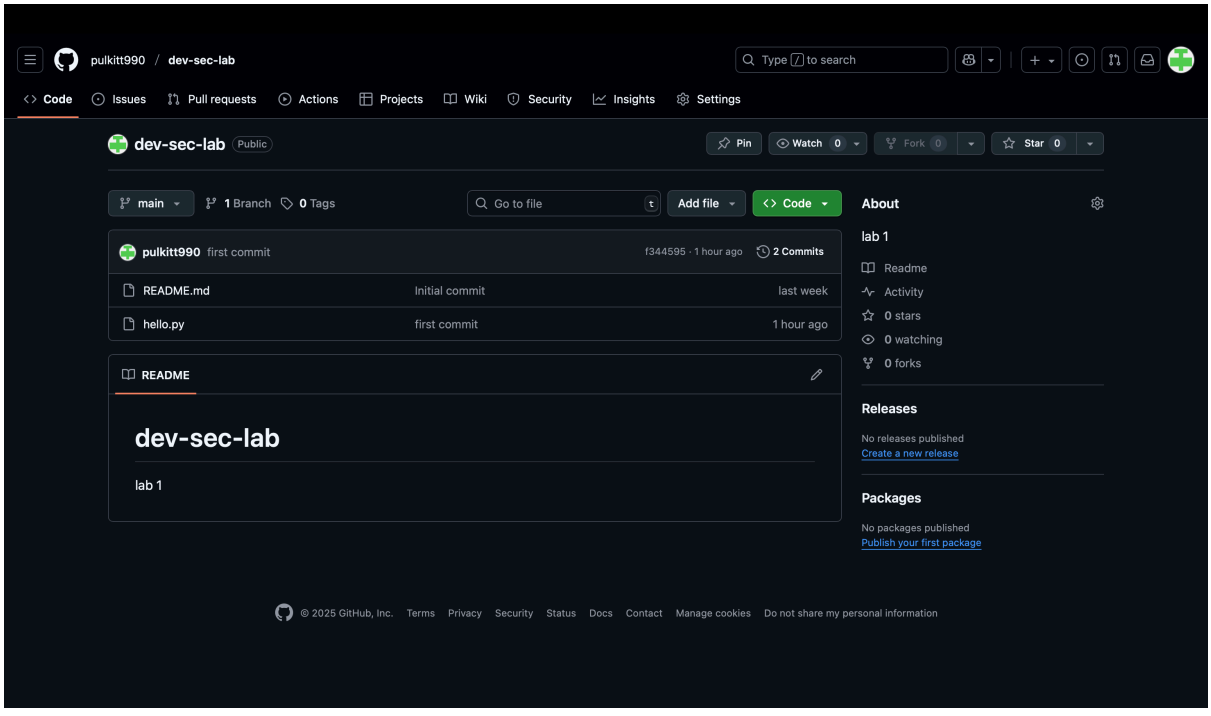
University of Petroleum and Energy Studies,

Experiment-1

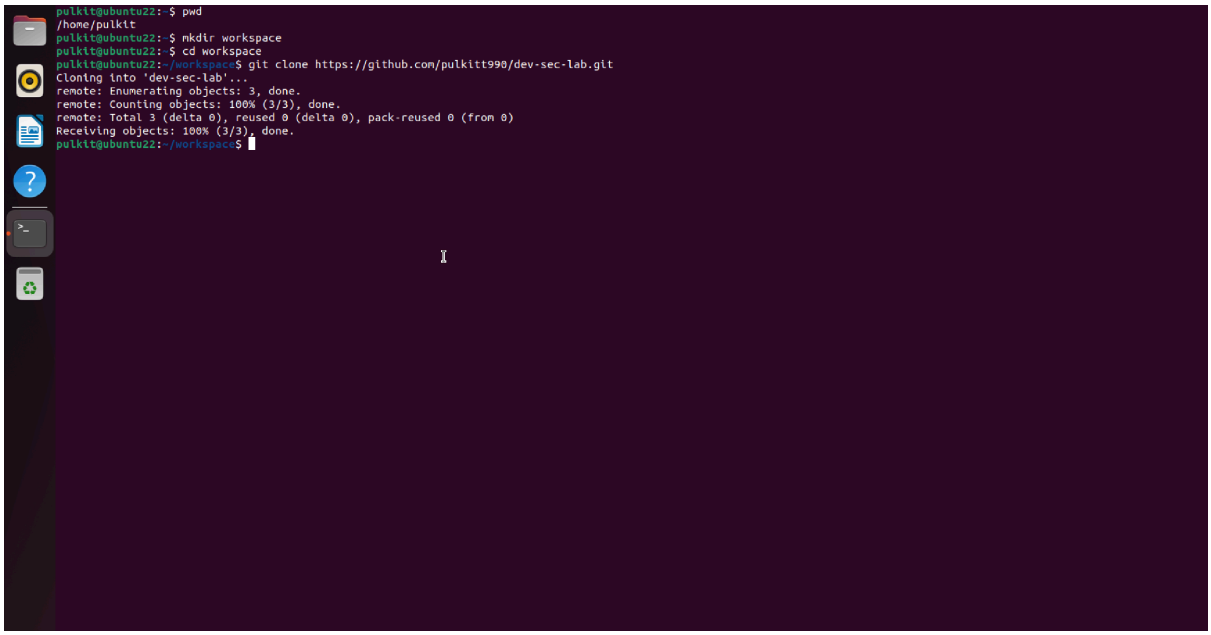
Setting up a git version control system.
Install git on your system

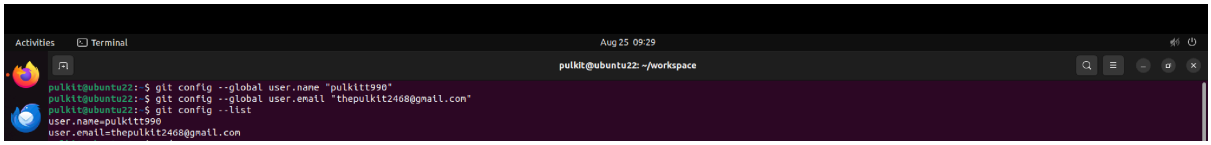


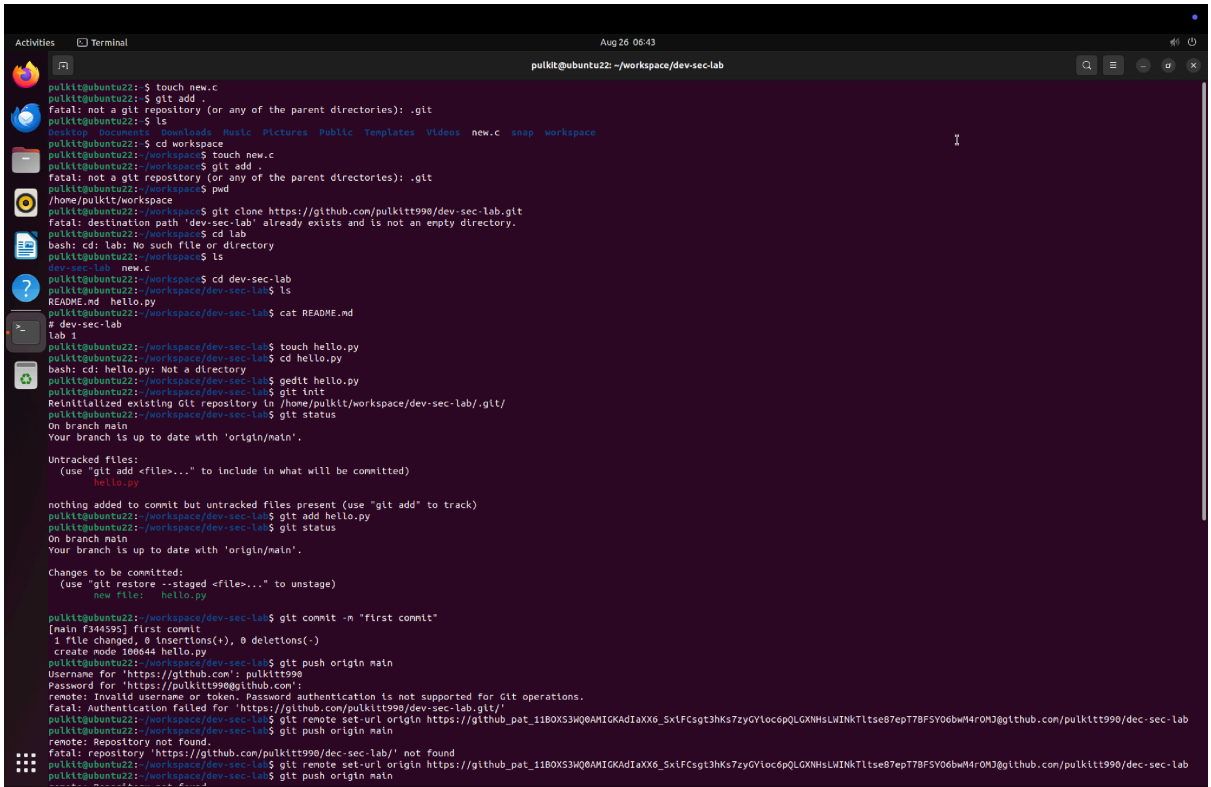
Create a new git repostory



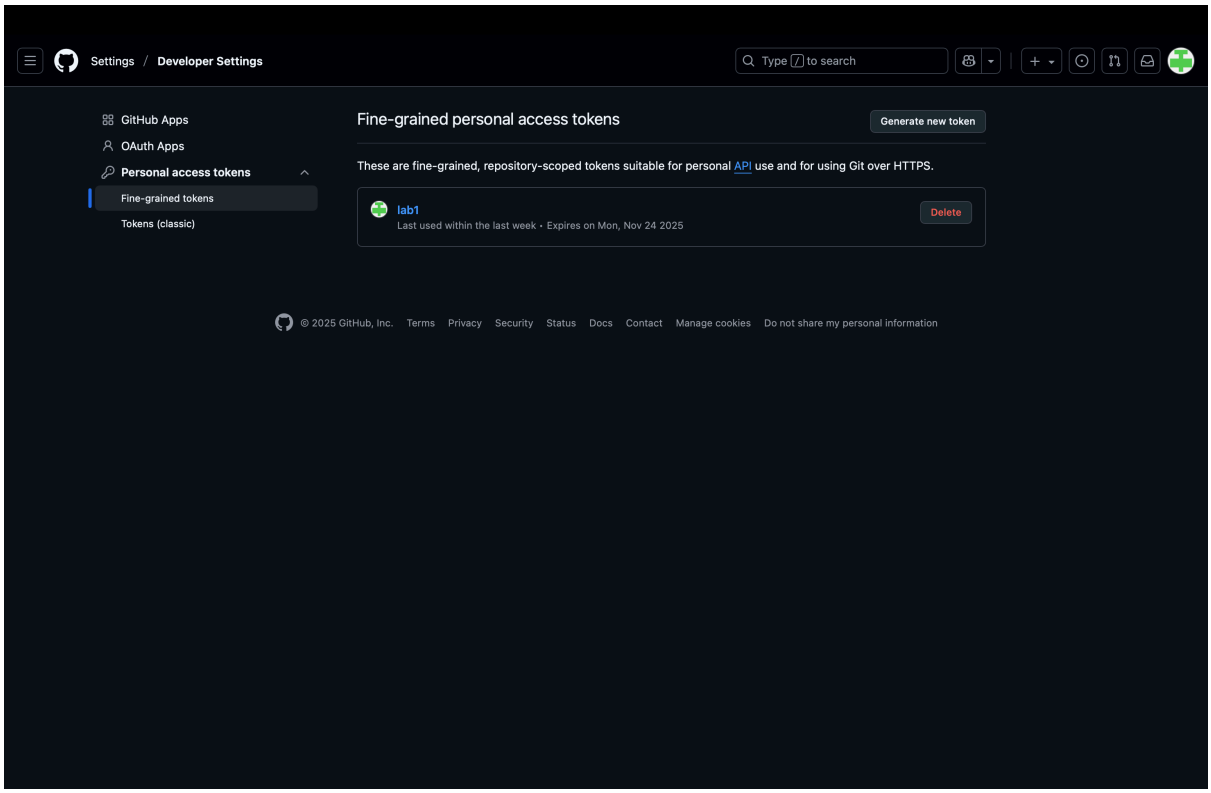
Add files,commit changes,and create branches



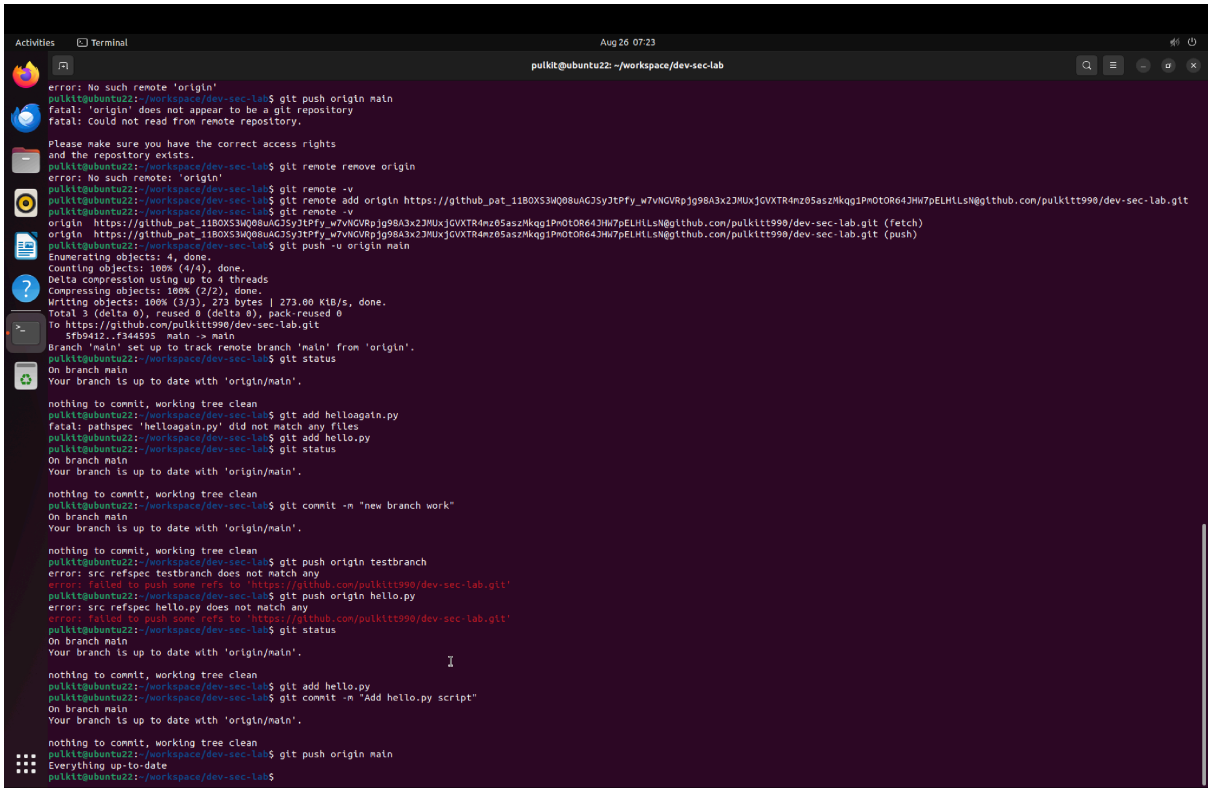


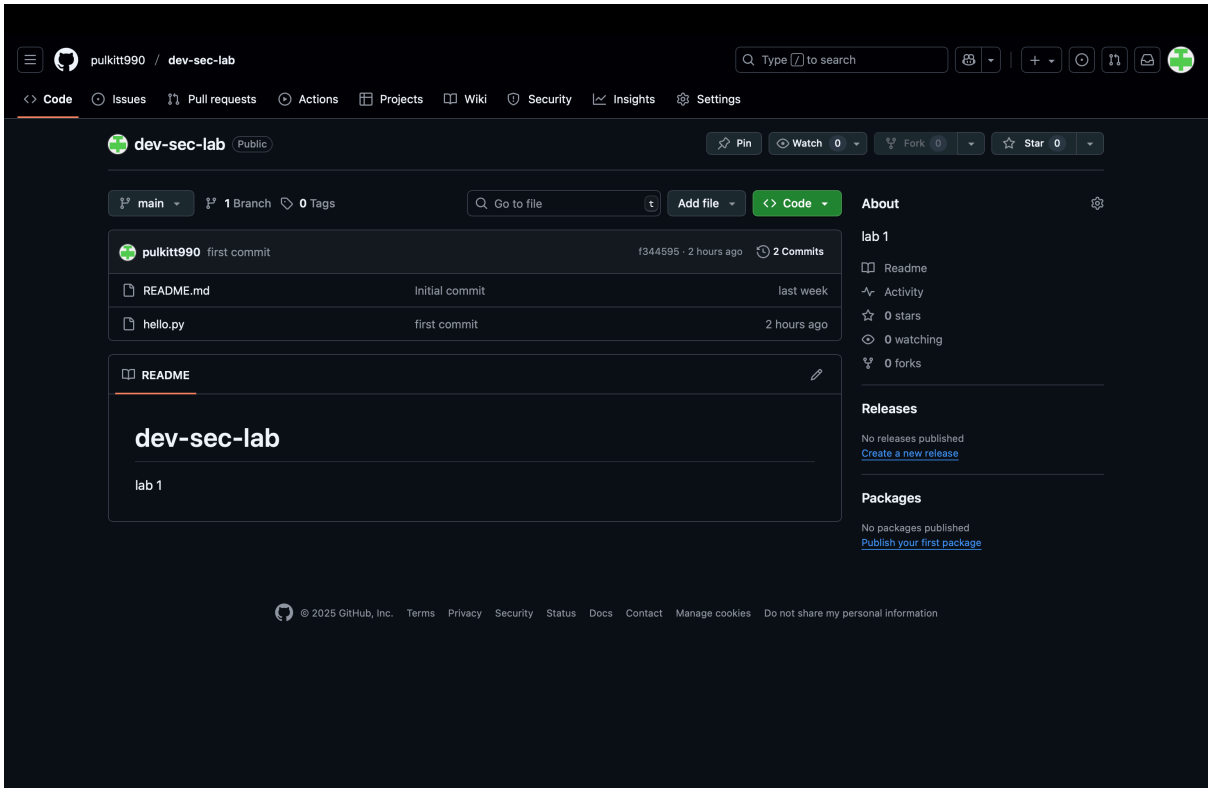


Genrate tokens



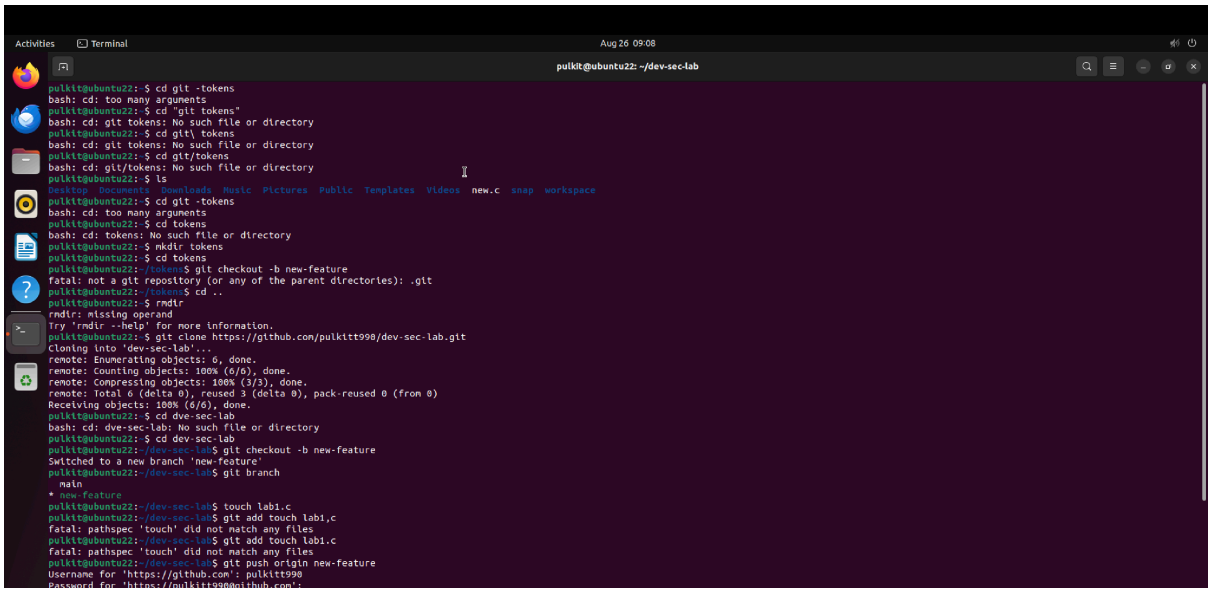
Push changes to a remote repository



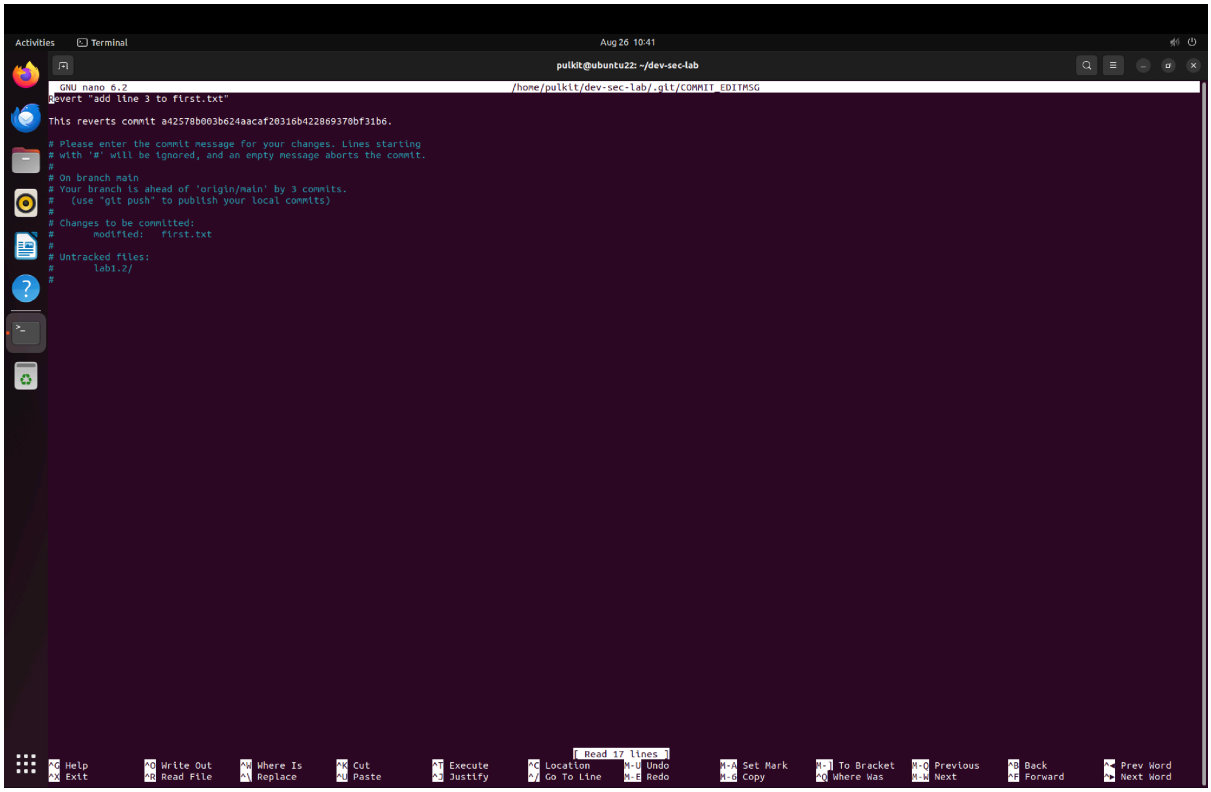


LAB 1.2 - Working with Git Revert

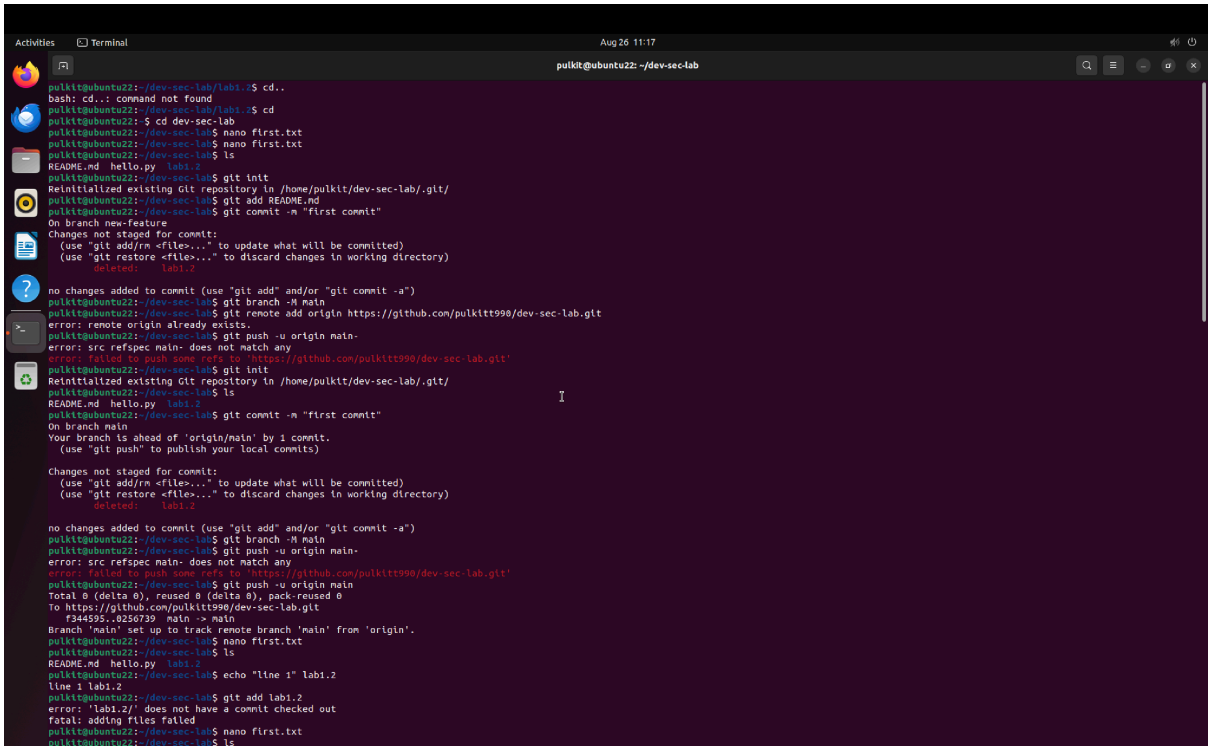
1.Set Up the Repository

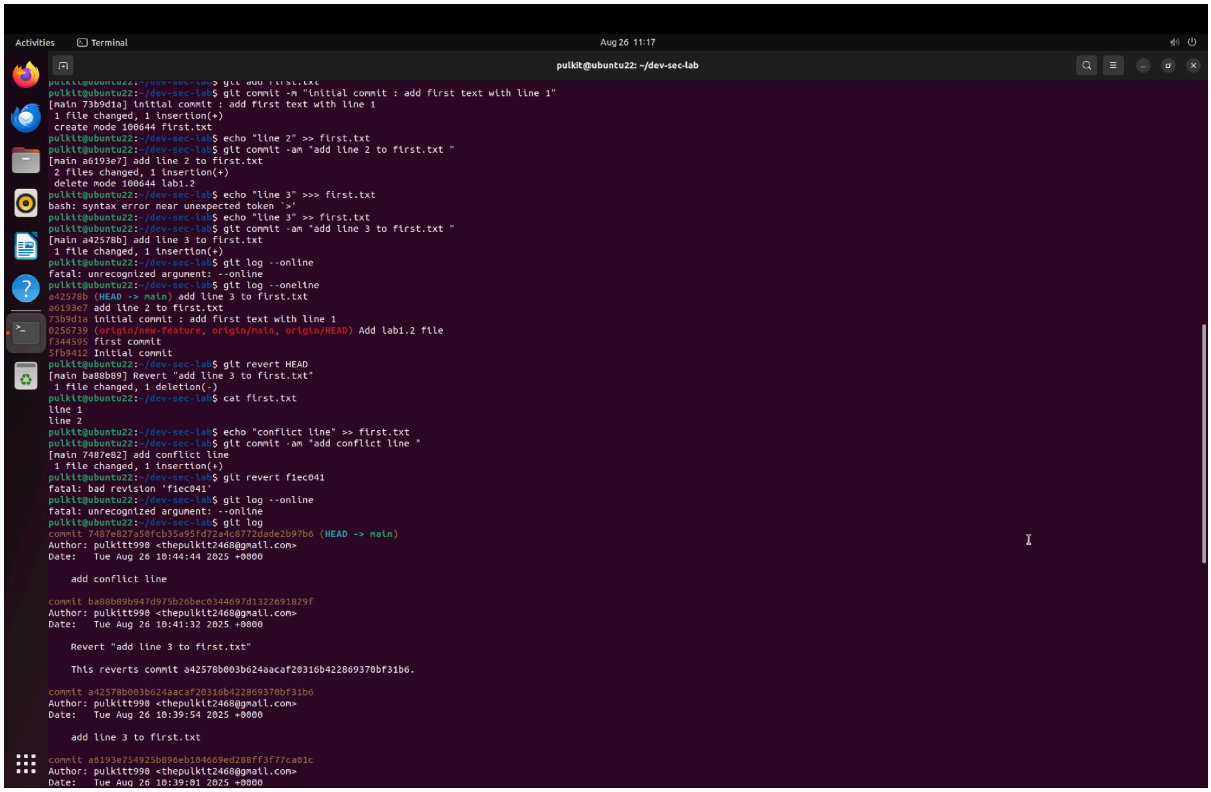


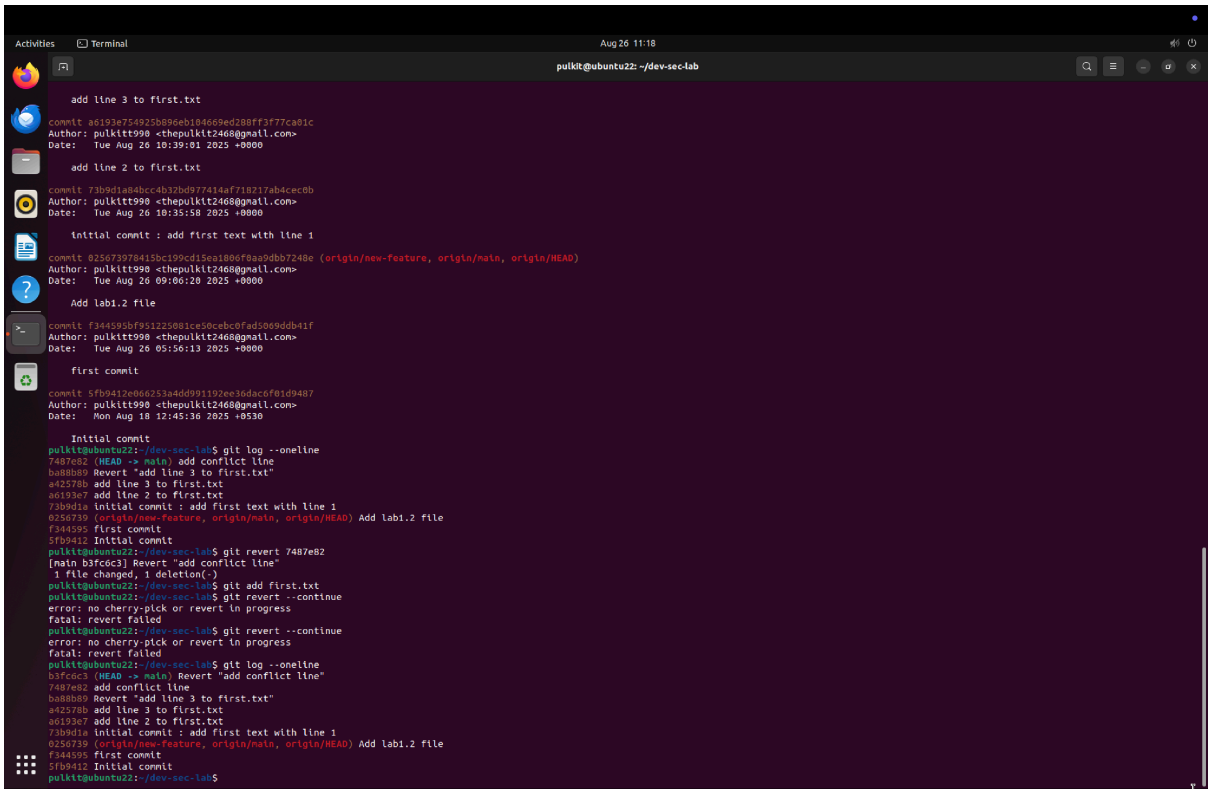
```
pulkit@ubuntu22:~/dev-sec-lab$ nano first.txt
pulkit@ubuntu22:~/dev-sec-lab$ ls
README.md  first.txt  hello.py  lab1.2
```



Handle Merge Conflicts During Revert :




Activities
Terminal
Aug 26 11:17
pulkit@ubuntu22: ~/dev-sec-lab
pulkit@ubuntu22:~/dev-sec-lab$ git commit -m "initial commit : add first text with line 1"
[main 73b9d1a] inittial commit : add first text with line 1
1 file changed, 1 insertion(+)
create mode 100644 first.txt
pulkit@ubuntu22:~/dev-sec-lab$ echo "line 2" >> first.txt
pulkit@ubuntu22:~/dev-sec-lab$ git commit -am "add line 2 to first.txt "
[main a6193e7] add line 2 to first.txt
2 files changed, 1 insertion(+)
delete mode 100644 lab1.2
pulkit@ubuntu22:~/dev-sec-lab$ echo "line 3" >>> first.txt
bash: syntax error near unexpected token `>'
pulkit@ubuntu22:~/dev-sec-lab$ echo "line 3" >> first.txt
pulkit@ubuntu22:~/dev-sec-lab$ git commit -am "add line 3 to first.txt "
[main a42578b] add line 3 to first.txt
1 file changed, 1 insertion(+)
pulkit@ubuntu22:~/dev-sec-lab$ git log --online
fatal: unrecognized argument: --online
pulkit@ubuntu22:~/dev-sec-lab$ git log --oneline
a42578b (HEAD -> main) add line 3 to first.txt
a6193e7 add line 2 to first.txt
73b9d1a initial commit : add first text with line 1
0256739 (origin/new-feature, origin/main, origin/HEAD) Add lab1.2 file
f344595 first commit
5fb9412 Initial commit
pulkit@ubuntu22:~/dev-sec-lab$ git revert HEAD
[main ba88b89] Revert "add line 3 to first.txt"
1 file changed, 1 deletion(-)
pulkit@ubuntu22:~/dev-sec-lab$ cat first.txt
line 1
line 2
pulkit@ubuntu22:~/dev-sec-lab$ echo "conflict line" >> first.txt
pulkit@ubuntu22:~/dev-sec-lab$ git commit -am "add conflict line "
[main 7487e82] add conflict line
1 file changed, 1 insertion(+)
pulkit@ubuntu22:~/dev-sec-lab$ git revert f1ec041
fatal: bad revision 'f1ec041'
pulkit@ubuntu22:~/dev-sec-lab$ git log --online
fatal: unrecognized argument: --online
pulkit@ubuntu22:~/dev-sec-lab$ git log
commit 7487e827a58fcb35a95fd72a4c8772dade2b97b6 (HEAD -> main)
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Tue Aug 26 10:44:44 2025 +0000
add conflict line
commit ba88b89b947d975b26bec0344697d1322691829f
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Tue Aug 26 10:41:32 2025 +0000
Revert "add line 3 to first.txt"
This reverts commit a42578b003b624aacaf20316b422869370bf31b6.
commit a42578b003b624aacaf20316b422869370bf31b6
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Tue Aug 26 10:39:54 2025 +0000
add line 3 to first.txt
commit a6193e754925b896eb104669ed288ff3f77ca01c
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Tue Aug 26 10:39:01 2025 +0000


Activities
Terminal
Aug 26 11:18
pulkit@ubuntu22: ~/dev-sec-lab
add line 3 to first.txt
commit a6193e754925b896eb104669ed288ff3f77ca01c
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Tue Aug 26 10:39:01 2025 +0000
add line 2 to first.txt
commit 73b9d1a84bcc4b32bd977414af718217ab4cec0b
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Tue Aug 26 10:35:58 2025 +0000
initial commit : add first text with line 1
commit 025673978415bc199cd15ea1806f8aa9dbb7248e (origin/new-feature, origin/main, origin/HEAD)
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Tue Aug 26 09:06:20 2025 +0000
Add lab1.2 file
commit f344595bf951225081ce50cebc8fad5069ddb41f
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Tue Aug 26 05:56:13 2025 +0000
first commit
commit 5fb9412e066253a4dd991192ee36dac6f01d9487
Author: pulkitt990 <thepulkit2468@gmail.com>
Date: Mon Aug 18 12:45:36 2025 +0530
Initial commit
pulkit@ubuntu22:~/dev-sec-lab$ git log --oneline
7487e82 (HEAD -> main) add conflict line
ba88b89 Revert "add line 3 to first.txt"
a42578b add line 3 to first.txt
a6193e7 add line 2 to first.txt
73b9d1a initial commit : add first text with line 1
0256739 (origin/new-feature, origin/main, origin/HEAD) Add lab1.2 file
f344595 first commit
5fb9412 Inttial commit
pulkit@ubuntu22:~/dev-sec-lab$ git revert 7487e82
[main b3fc6c3] Revert "add conflict line"
1 file changed, 1 deletion(-)
pulkit@ubuntu22:~/dev-sec-lab$ git add first.txt
pulkit@ubuntu22:~/dev-sec-lab$ git revert --continue
error: no cherry-pick or revert in progress
fatal: revert failed
pulkit@ubuntu22:~/dev-sec-lab$ git revert --continue
error: no cherry-pick or revert in progress
fatal: revert failed
pulkit@ubuntu22:~/dev-sec-lab$ git log --oneline
b3fc6c3 (HEAD -> main) Revert "add conflict line"
7487e82 add conflict line
ba88b89 Revert "add line 3 to first.txt"
a42578b add line 3 to first.txt
a6193e7 add line 2 to first.txt
73b9d1a initial commit : add first text with line 1
0256739 (origin/new-feature, origin/main, origin/HEAD) Add lab1.2 file
f344595 first commit
5fb9412 Initial commit
pulkit@ubuntu22:~/dev-sec-lab$

Lab Exercise 2- Working with Git Reset

Name: Pulkit

Sap ID: 500125835

Batch –

Lab Exercise: Git Reset

This lab exercise will guide you through the usage of the git reset command in various scenarios. The git reset command is used to undo changes in the Git history, working directory, or staging area. There are three main modes: **soft**, **mixed**, and **hard**.

Objective

- Learn how to use git reset to modify the commit history, unstage files, or discard changes.
- Understand the differences between --soft, --mixed, and --hard reset modes.

Prerequisites

1. Install Git on your system.
2. Set up a Git repository:

```
git init git-reset-lab

cd git-reset-lab
```

Steps

1. Set Up the Repository

1. Create and commit an initial file:

```
echo "Line 1" > file.txt

git add file.txt

git commit -m "Initial commit: Add Line 1"
```

2. Add a second change:

```
echo "Line 2" >> file.txt

git commit -am "Add Line 2"
```

3. Add a third change:

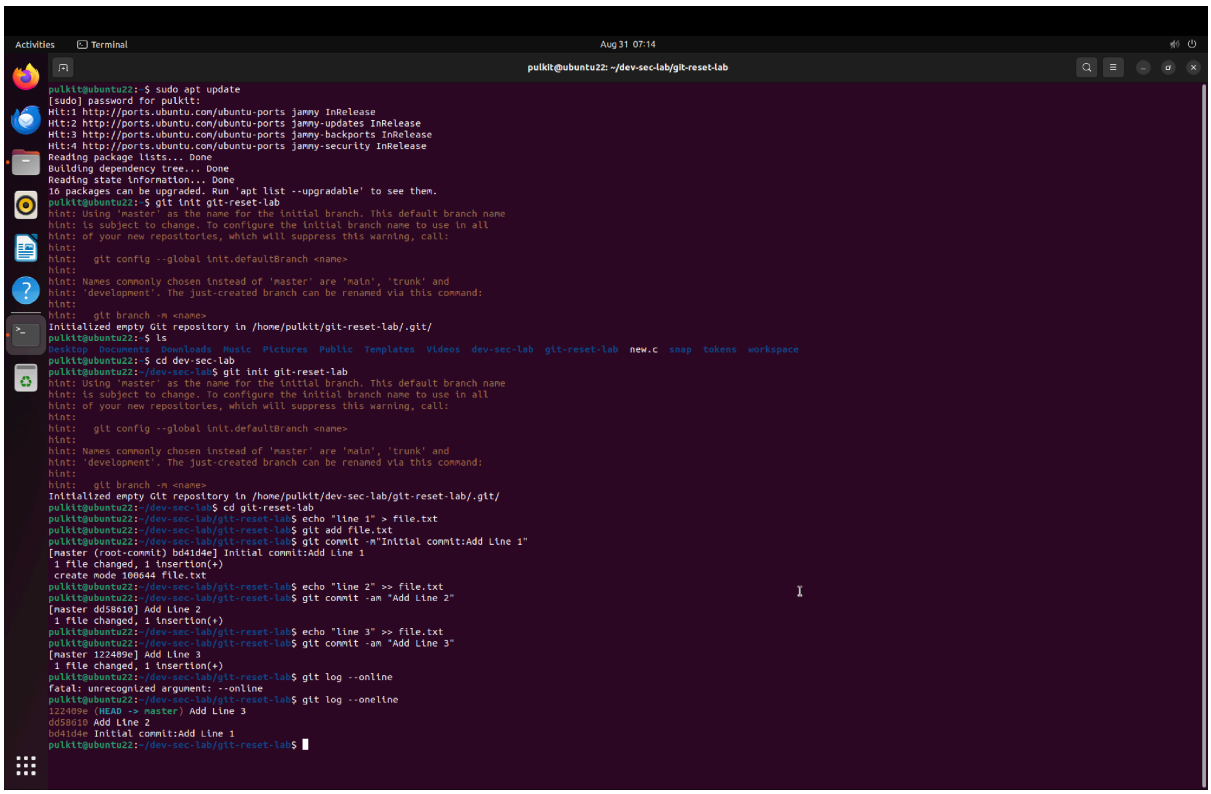
```
echo "Line 3" >> file.txt

git commit -am "Add Line 3"
```

4. Check the commit history:

```
git log --oneline
```

Output:



---

2. Use git reset --soft

This mode moves the HEAD pointer to an earlier commit but keeps the changes in the staging area.

1. Reset to the second commit:

```
git reset --soft HEAD~1
```

2. Check the commit history:

```
git log --oneline
```

3. Verify the staged changes:

```
git status
```

Output:

4. If needed, re-commit the changes:

```
git commit -m "Recommit Line 3"
```

---

3. Use git reset --mixed

This mode moves the HEAD pointer and unstages the changes but keeps them in the working directory.

1. Reset to the first commit:

```
git reset --mixed HEAD~1
```

2. Check the commit history:

```
git log --oneline
```

3. Verify the changes in the working directory:

```
git status
```

Output:

4. If needed, stage and re-commit:

```
git add file.txt

git commit -m "Recommit Line 2 and Line 3"
```

---

4. Use git reset --hard

This mode moves the HEAD pointer and discards all changes in the staging area and working directory.

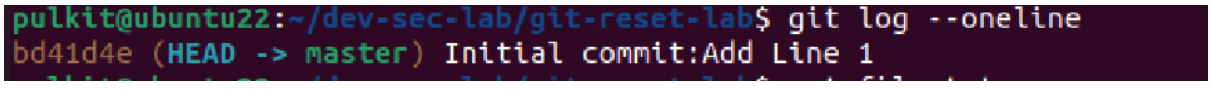
1. Reset to the initial commit:

```
git reset --hard HEAD~1
```

2. Check the commit history:

```
git log --oneline
```

Output:



3. Verify the working directory:

```
cat file.txt
```

Output:

---

5. Use git reset with a Commit Hash

1. Add some changes for demonstration:

```
echo "Line 2" >> file.txt

git commit -am "Add Line 2"

echo "Line 3" >> file.txt

git commit -am "Add Line 3"
```

2. Get the commit hash for the initial commit:

```
git log --oneline
```

3. Reset to the initial commit using the hash:

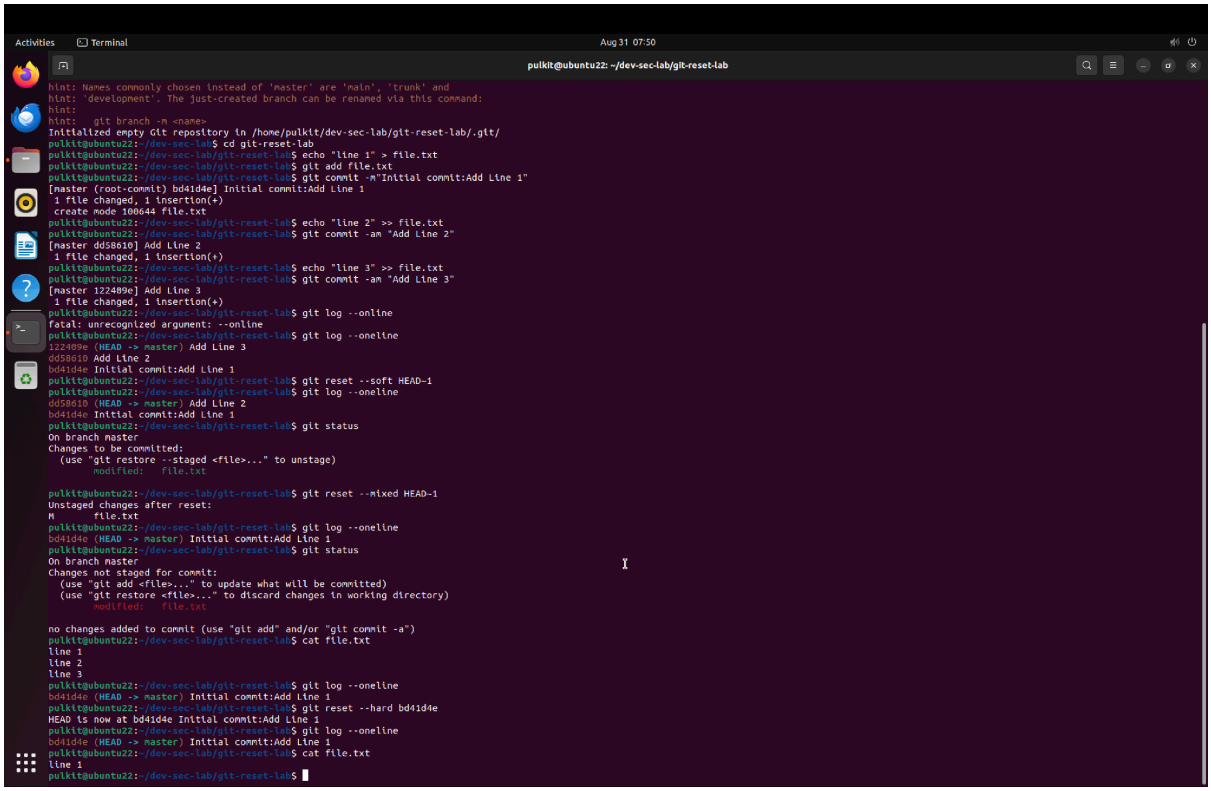
```
git reset --hard <commit-hash>
```

4. Verify the working directory and commit history:

```
git log --oneline

cat file.txt
```

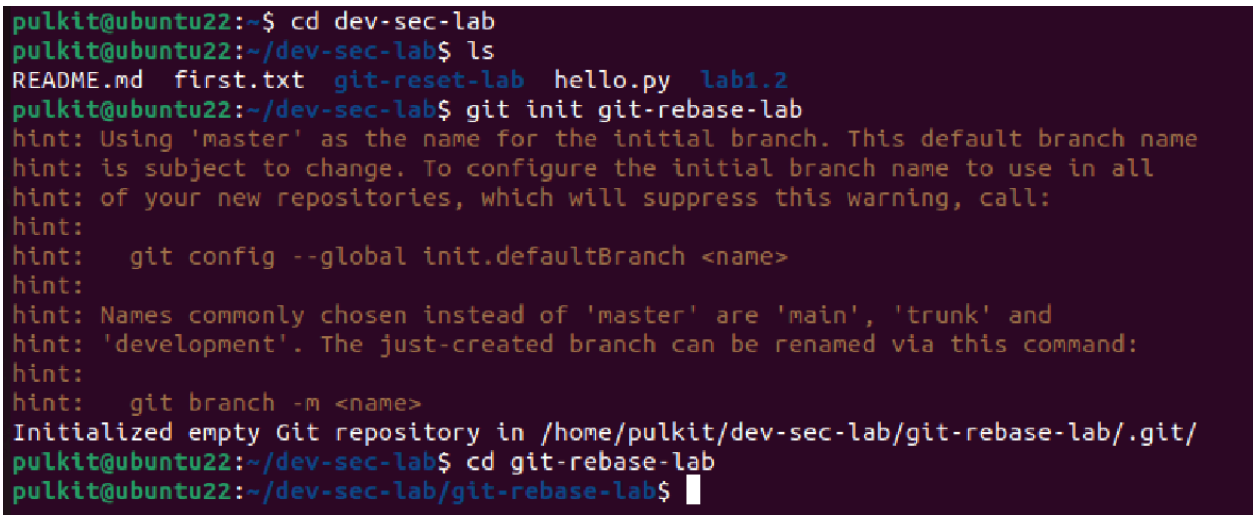
Output:



Summary of Commands

Mode	Effect	Command Example
--soft	Moves HEAD, keeps changes staged.	git reset --soft HEAD~1
--mixed	Moves HEAD, unstages changes, keeps them in working dir.	git reset --mixed HEAD~1
--hard	Moves HEAD, discards all changes in staging and working dir.	git reset --hard HEAD~1

Difference btw revert and reset


pulkit@ubuntu22:~$ cd dev-sec-lab
pulkit@ubuntu22:~/dev-sec-lab$ ls
README.md first.txt git-reset-lab hello.py lab1.2
pulkit@ubuntu22:~/dev-sec-lab$ git init git-rebase-lab
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint: git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint: git branch -m <name>
Initialized empty Git repository in /home/pulkit/dev-sec-lab/git-rebase-lab/.git/
pulkit@ubuntu22:~/dev-sec-lab$ cd git-rebase-lab
pulkit@ubuntu22:~/dev-sec-lab/git-rebase-lab$

Lab Exercise 3- Working with Git Rebase

Name: Pulkit

Sap ID: 500125835

Batch - 3

Lab Exercise: Git Rebase

This exercise demonstrates the use of git rebase in a scenario where no conflicts occur.

Objective

1. Learn how to rebase branches when there are no conflicting changes.
2. Understand the clean, linear history created by git rebase.

Prerequisites

1. Install Git on your system.
2. Initialize a Git repository:

Step-by-Step Instructions

1. Set Up the Repository

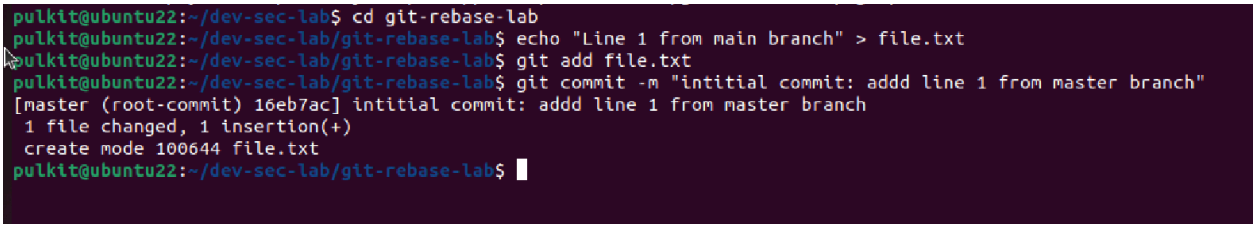
1. Create the main branch and make the initial commit:

```
echo "Line 1 from main branch" > file.txt

git add file.txt
```

git commit -m "Initial commit: Add Line 1 from main branch"

2. Create a new branch feature-branch:



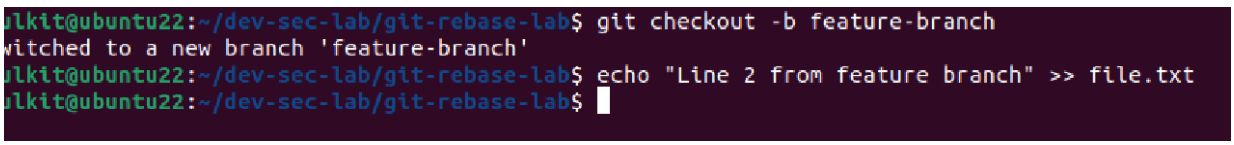

```
git checkout -b feature-branch
```

3. Add a new line to file.txt in feature-branch:

```
echo "Line 2 from feature branch" >> file.txt

git add file.txt

git commit -m "Add Line 2 from feature branch"
```



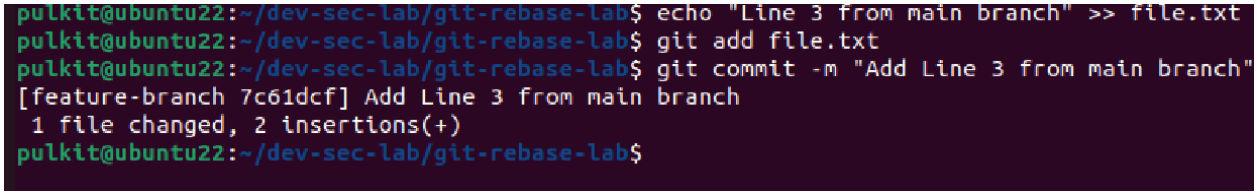
4. Switch back to the main branch and add another line:

```
git checkout main

echo "Line 3 from main branch" >> file.txt

git add file.txt

git commit -m "Add Line 3 from main branch"
```



---

2. Rebase feature-branch onto main

1. Switch to feature-branch:

```
git checkout feature-branch
```

2. Rebase feature-branch onto main:

```
git rebase main
```

3. Git will replay the commit from feature-branch onto the main branch. Since there are no conflicts, the rebase completes automatically.

3. Verify the Rebase

1. View the commit history:

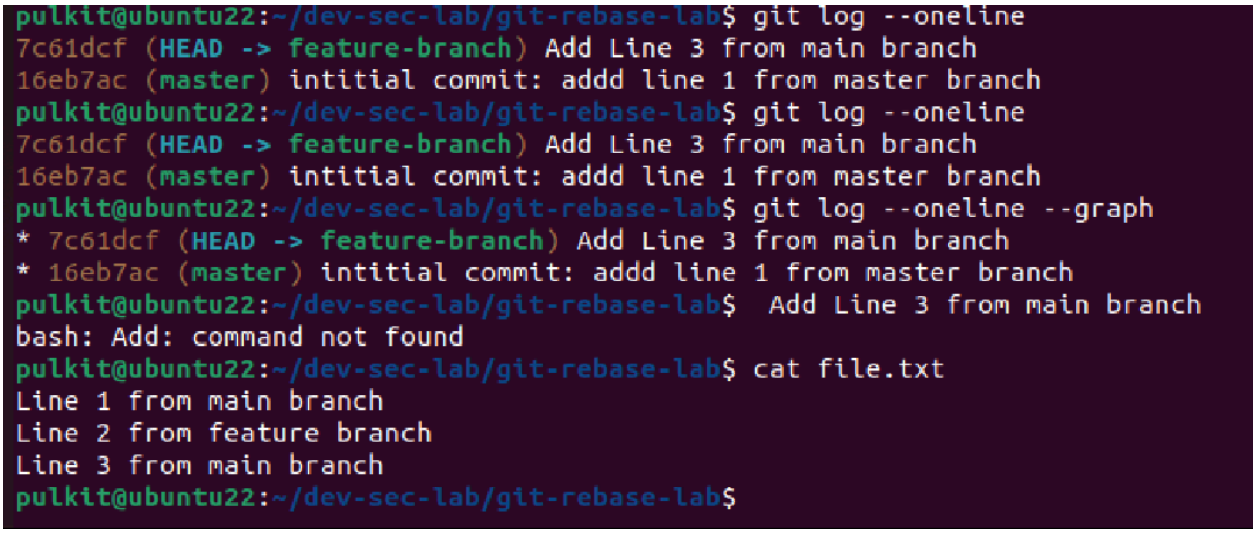
```
git log --oneline

git log --oneline --graph
```

2. Check the contents of file.txt:

```
cat file.txt
```

Output:



Lab Exercise 4- Signed Commits in Git and GitHub

Objective:

To configure Git to sign commits with GPG, push them to GitHub, and verify commit authenticity for secure code contribution.

Prerequisites:

- Git installed on your system
- GPG (GNU Privacy Guard) installed and configured
- GitHub account with a repository (you own or have write access to)
- Basic knowledge of Git commands

Step 1 – Generate or Use an Existing GPG Key

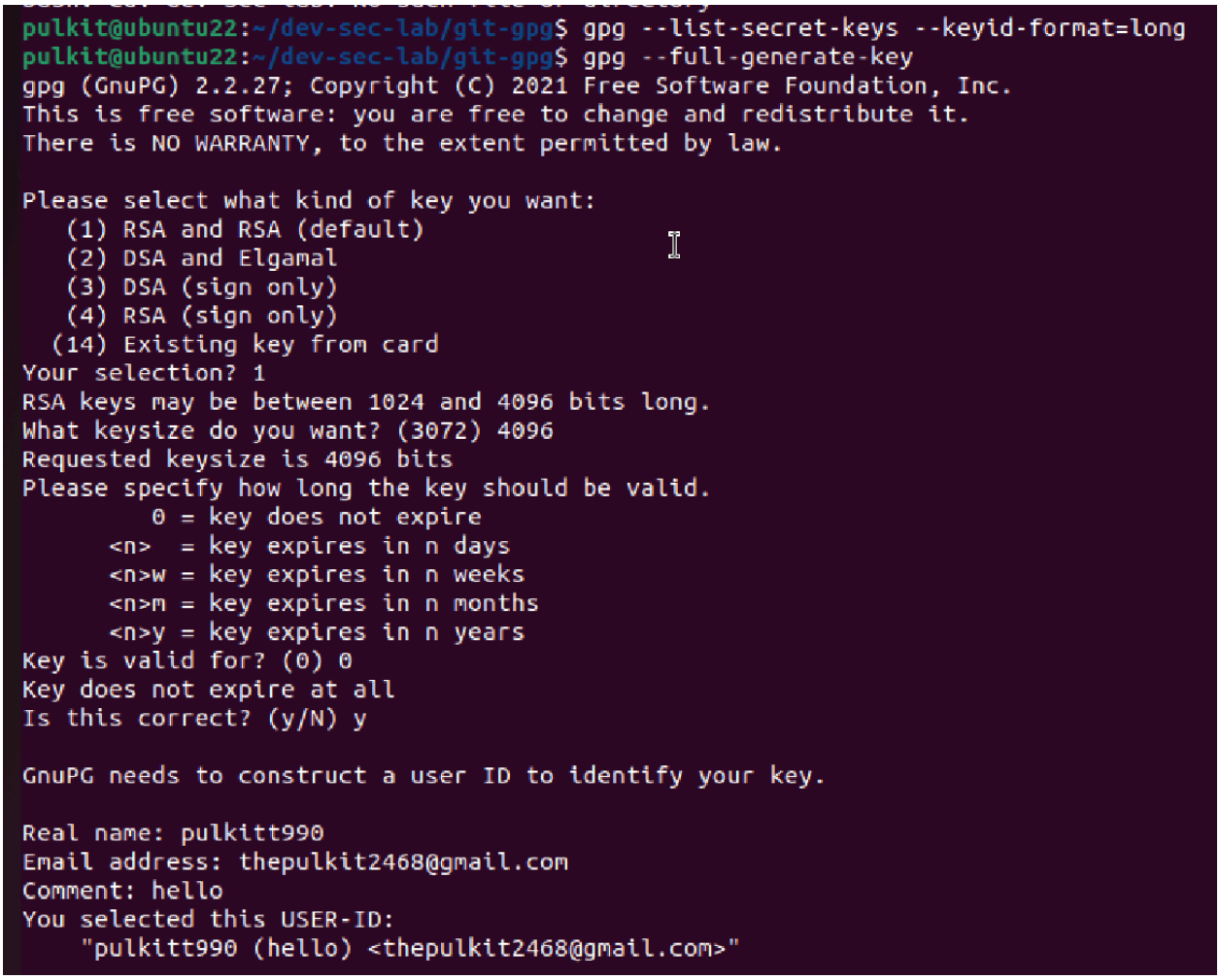
1. **Check for existing keys**

```
gpg --list-secret-keys --keyid-format=long
```

2. **If no key exists, generate a new one**

```
gpg --full-generate-key
```

- Select **RSA and RSA**
- Key size: **4096**
- Expiration: **0** (never) or a fixed date
- Enter your **GitHub-registered name and email**



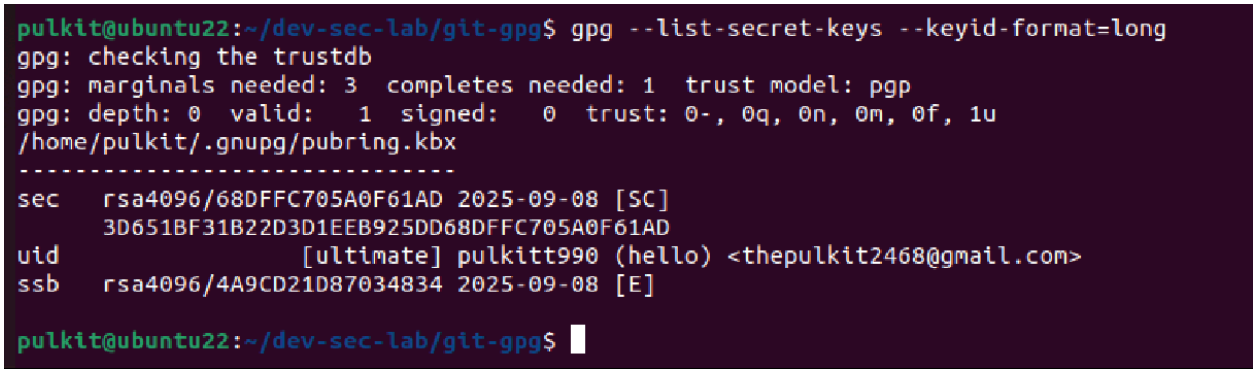
3. **Get your key ID**

```
gpg --list-secret-keys --keyid-format=long
```

Example output:

```
sec   rsa4096/3AA5C34371567BD2 2025-08-13 [SC]
```

Here, 3AA5C34371567BD2 is your key ID.



Step 2 – Add GPG Key to GitHub

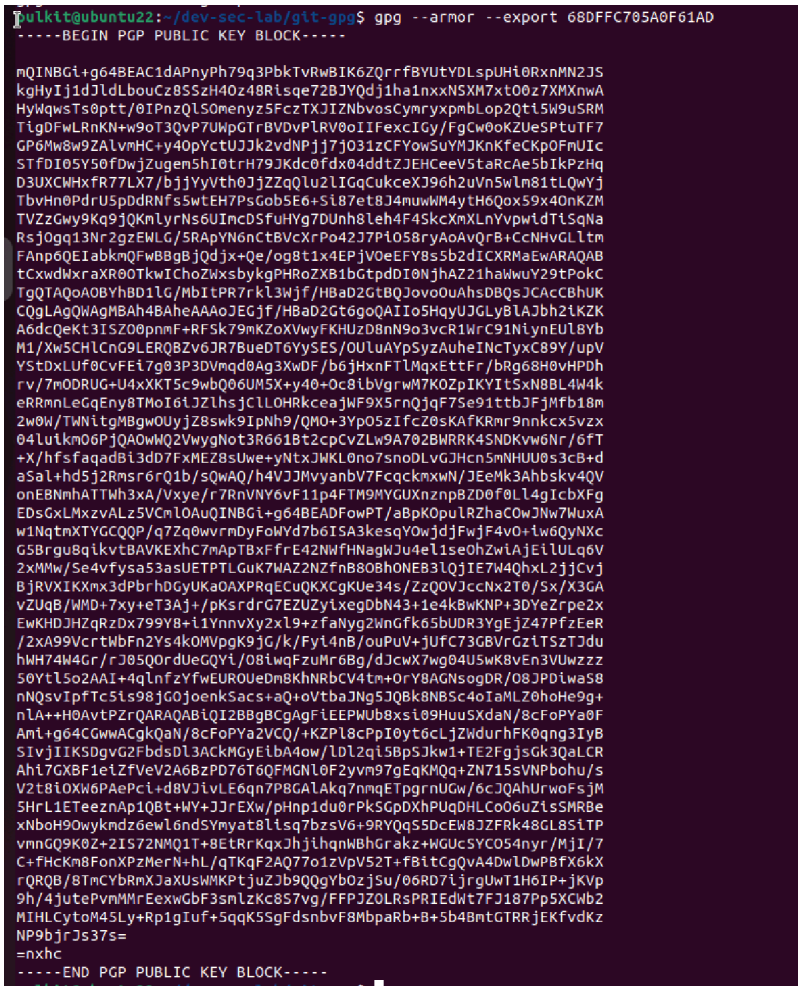
1. Export your public key:

```
gpg --armor --export YOUR_KEY_ID
```

2. Copy the output.

3. Go to **GitHub → Settings → SSH and GPG Keys → New GPG Key**.

4. Paste your key and save.



Step 3 – Configure Git for Signed Commits

1. Tell Git which key to use:

```
git config --global user.signingkey YOUR_KEY_ID
```

2. Enable signing for all commits:

```
git config --global commit.gpgsign true
```

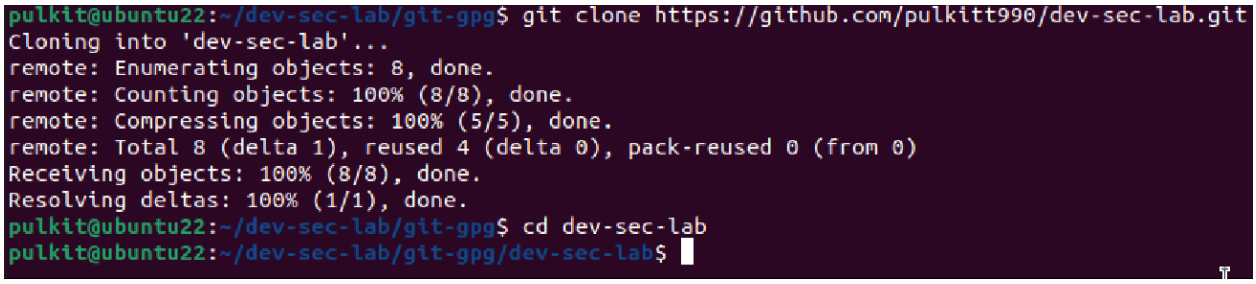
```
pulkit@ubuntu22:~/dev-sec-lab/git-gpg$ git config --global user.signingkey 68DFFC705A0F61AD
pulkit@ubuntu22:~/dev-sec-lab/git-gpg$ git config --global commit.gpgsign true
pulkit@ubuntu22:~/dev-sec-lab/git-gpg$
```

Step 4 – Make a Signed Commit

1. Clone your repo (or use an existing one):

```
git clone https://github.com/<username>/<repository>.git

cd <repository>
```



2. Edit or create a file:

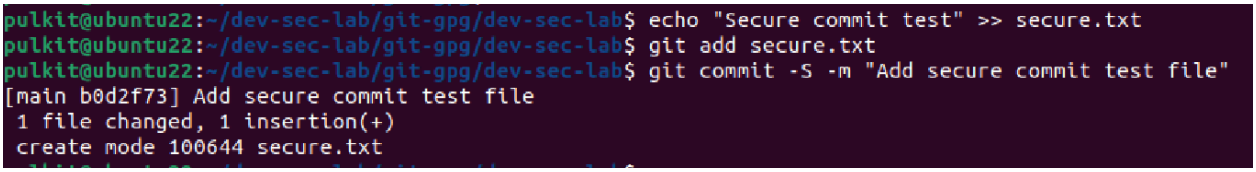
```
echo "Secure commit test" >> secure.txt

git add secure.txt
```

3. Commit with signing:

```
git commit -S -m "Add secure commit test file"
```

4. Enter your GPG passphrase when prompted.



Step 5 – Push and Verify on GitHub

1. Push the commit:

```
git push origin main
```

2. Go to your repository on GitHub → Click the commit → You should see a **green "Verified" badge**.

```
pulkit@ubuntu22:~/dev-sec-lab/git-gpg/dev-sec-lab$ git push -u origin main
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 960 bytes | 960.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/pulkitt990/dev-sec-lab.git
   0256739..b0d2f73  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
pulkit@ubuntu22:~/dev-sec-lab/git-gpg/dev-sec-lab$
```

Step 6 – Local Verification of Commit

```
git log --show-signature
```

This will display the GPG verification details locally.

```
pulkit@ubuntu22:~/dev-sec-lab/git-gpg/dev-sec-lab$ git log --show-signature
commit b0d2f73ac586b98e22a0bf5faaf039df9e52eb87 (HEAD -> main, origin/main, origin/HEAD)
gpg: Signature made Mon Sep  8 07:40:57 2025 UTC
gpg:                using RSA key 3D651BF31B22D3D1EEB925DD68DFFC705A0F61AD
gpg: Good signature from "pulkitt990 (hello) <thepulkit2468@gmail.com>" [ultimate]
Author: pulkitt990 <thepulkit2468@gmail.com>
Date:   Mon Sep 8 07:40:56 2025 +0000

    Add secure commit test file

commit 025673978415bc199cd15ea1806f0aa9dbb7248e (origin/new-feature)
Author: pulkitt990 <thepulkit2468@gmail.com>
Date:   Tue Aug 26 09:06:20 2025 +0000

    Add lab1.2 file

commit f344595bf951225081ce50cebc0fad5069ddb41f
Author: pulkitt990 <thepulkit2468@gmail.com>
Date:   Tue Aug 26 05:56:13 2025 +0000

    first commit

commit 5fb9412e066253a4dd991192ee36dac6f01d9487
gpg: Signature made Mon Aug 18 07:15:36 2025 UTC
gpg:                using RSA key B5690EEEBB952194
gpg: Can't check signature: No public key
Author: pulkitt990 <thepulkit2468@gmail.com>
Date:   Mon Aug 18 12:45:36 2025 +0530

    Initial commit
```

Use Case

Signed commits prevent identity spoofing in collaborative projects, ensuring only verified authors can make trusted changes in critical codebases.

Lab Exercise 5- Generate and Use SSH Key with Git and GitHub

Objective:

To learn how to generate an SSH key, add it to GitHub, and use it to securely connect and push code without repeatedly entering a password.

Prerequisites

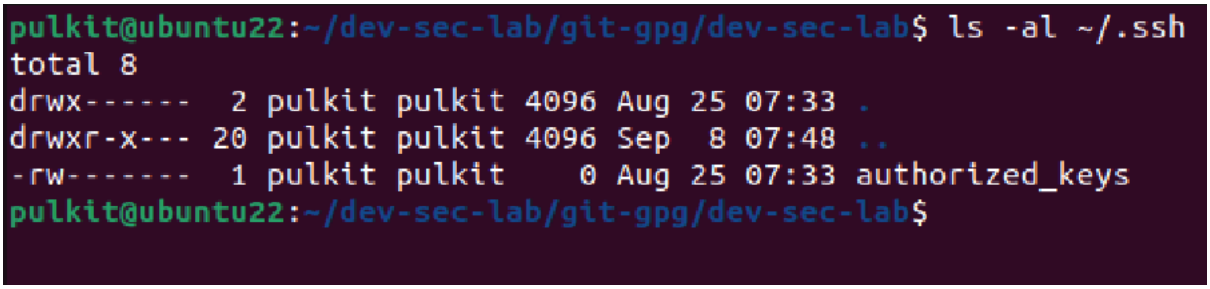
- Git installed on your local machine
- GitHub account
- Basic understanding of Git commands

Step 1 – Check for Existing SSH Keys

Run:

```
ls -al ~/.ssh
```

Look for files like id_rsa and id_rsa.pub. If they exist, you may already have an SSH key.



Step 2 – Generate a New SSH Key

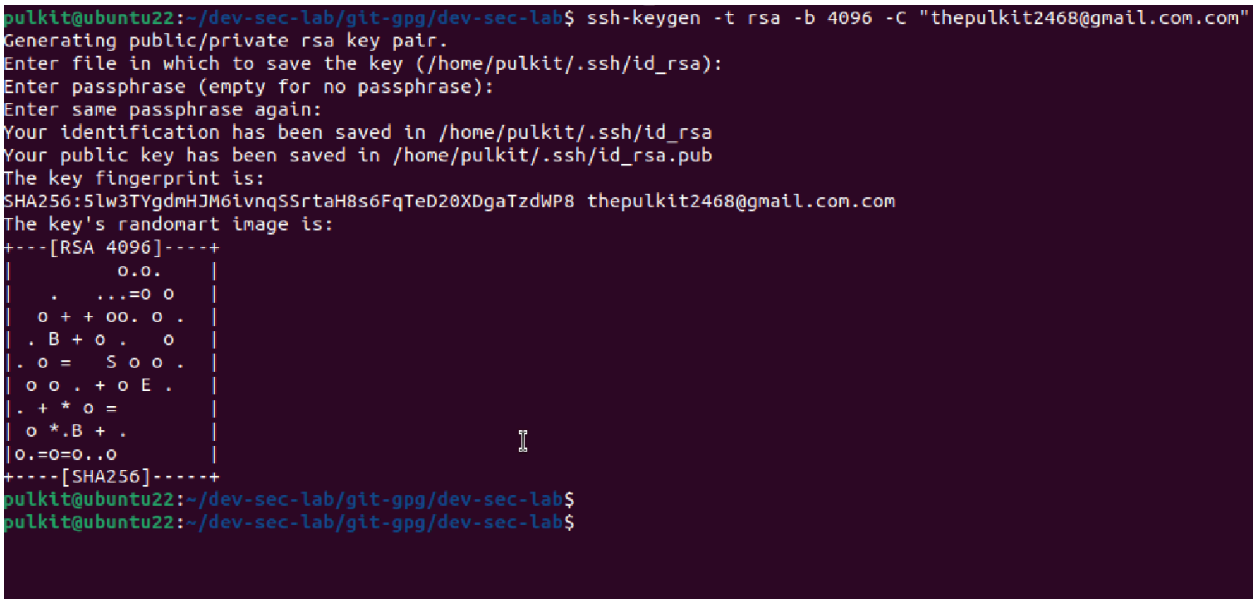
Run:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

- **-t rsa** → key type
- **-b 4096** → key length
- **-C** → comment (your GitHub email)

When prompted:

- Press **Enter** to save in the default location: /home/user/.ssh/id_rsa (Linux/ Mac) or C:\Users\<username>\.ssh\id_rsa (Windows)
- Optionally, set a passphrase for extra security.



Step 3 – Start the SSH Agent

```
eval "$(ssh-agent -s)"
```

Step 4 – Add SSH Key to the Agent

```
ssh-add ~/.ssh/id_rsa
```

Step 5 – Add SSH Key to GitHub

1. Copy the public key:

```
cat ~/.ssh/id_rsa.pub
```

2. Log in to GitHub → **Settings** → **SSH and GPG Keys** → **New SSH key**.

3. Paste the key and save.

Step 6 – Test SSH Connection

```
ssh -T git@github.com
```

Expected output:

```
Hi <username>! You've successfully authenticated, but GitHub does not provide shell access.
```

Step 7 – Use SSH to Clone a Repository

```
git clone git@github.com:<username>/<repository>.git
```

Now you can pull and push without entering your username/password.

```
pulkit@ubuntu22:~/dev-sec-lab/git-gpg/dev-sec-lab$ git clone https://github.com/pulkitt990/dev-sec-lab.git
Cloning into 'dev-sec-lab'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 11 (delta 2), reused 7 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (11/11), done.
Resolving deltas: 100% (2/2), done.
pulkit@ubuntu22:~/dev-sec-lab/git-gpg/dev-sec-lab$
```

Use Case

Scenario:

An organization’s developers often need to push code to GitHub multiple times a day.

Using SSH keys eliminates the need to repeatedly enter credentials, while maintaining secure, encrypted communication between the developer's machine and GitHub.

Table – HTTPS vs SSH for GitHub

Feature	**HTTPS**	**SSH**
Authentication	Username & password / token	SSH key pair
Convenience	Requires login each session	No password once key is added
Security	Encrypted, but password-based auth	Encrypted, key-based authentication
Best For	Occasional access	Frequent development work