Git and GitHub Lab Guide

Lab 1: Installing Git

Objective: Learn how to install Git and verify the installation.

Steps:

Windows:

1. Download Git from https://git-scm.com/download/win.
2. Run the installer, keeping default settings unless you need to change them.
3. After installation, open Git Bash from the Start menu.

macOS:
Install Git using Homebrew:
`brew install git` or download Git from https://git-scm.com/download/mac and follow the installation instructions.

Linux:
Ubuntu/Debian:
```
sudo apt update
sudo apt install git
```

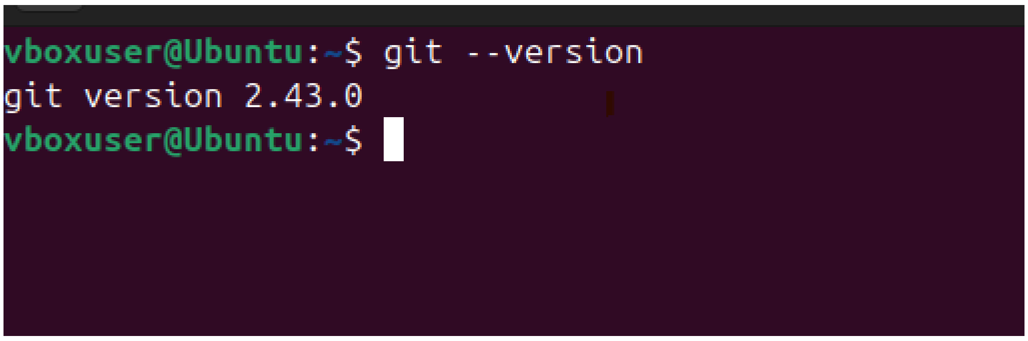
Fedora:
```
sudo dnf install git
```

CentOS/RHEL:
```
sudo yum install git
```

Verification:

Check the Git version to ensure it is installed:

```
git --version
```



```
vboxuser@Ubuntu:~$ git --version
git version 2.43.0
vboxuser@Ubuntu:~$ 
```

Lab 2: Configuring Git

Objective: Configure Git for your local system by setting up your name and email address. Git uses these settings to identify the author of commits.

Steps:

Set your username:

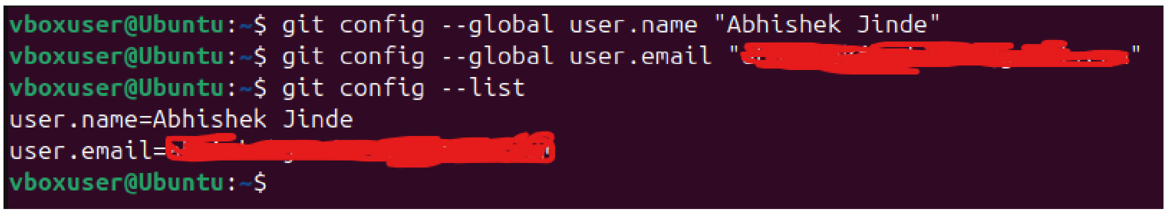
git config --global user.name "Your Name"

Set your email address:

git config --global user.email "your.email@example.com"

Verify your configuration:

git config --list



```
vboxuser@Ubuntu:~$ git config --global user.name "Abhishek Jinde"
vboxuser@Ubuntu:~$ git config --global user.email "[redacted]"
vboxuser@Ubuntu:~$ git config --list
user.name=Abhishek Jinde
user.email=[redacted]
vboxuser@Ubuntu:~$
```

Lab 3: Initializing a Repository

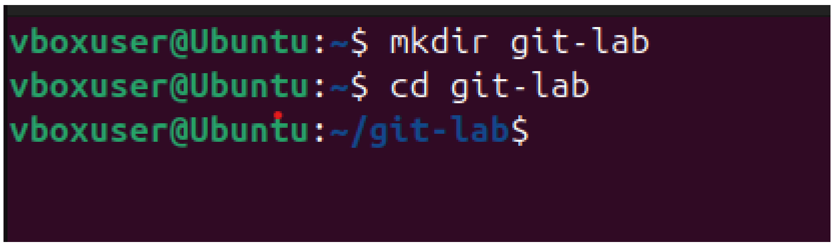
Objective: Create a new Git repository from scratch, add files, and commit them.

Steps:

1. **Create a directory for your project:**

   mkdir git-lab

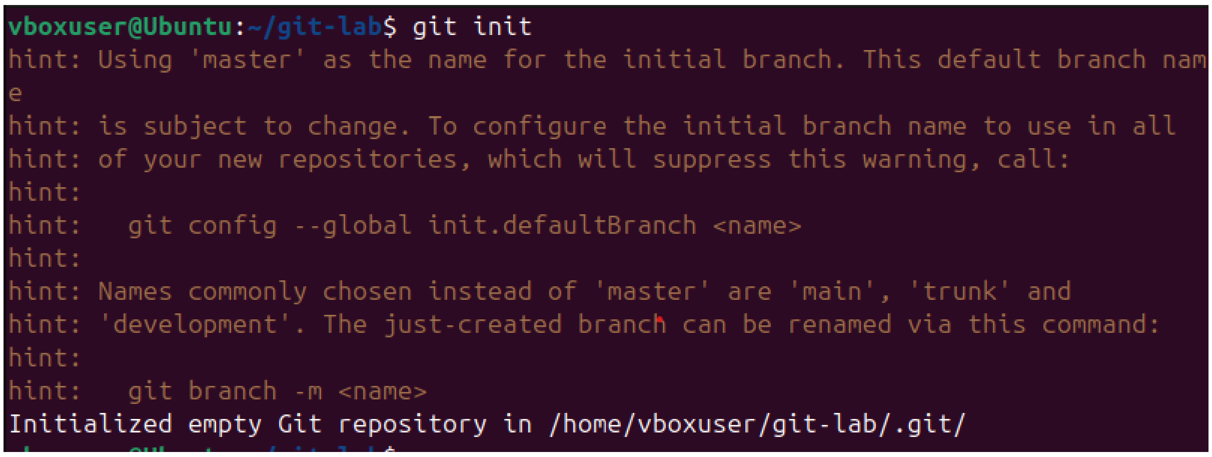
   cd git-lab



```
vboxuser@Ubuntu:~$ mkdir git-lab
vboxuser@Ubuntu:~$ cd git-lab
vboxuser@Ubuntu:~/git-lab$
```

2. **Initialize a Git repository :**

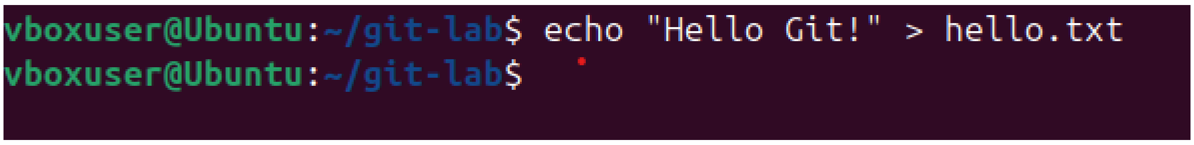
   git init



```
vboxuser@Ubuntu:~/git-lab$ git init
hint: Using 'master' as the name for the initial branch. This default branch nam
e
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/vboxuser/git-lab/.git/
```

3. **Create a file in the repository:**

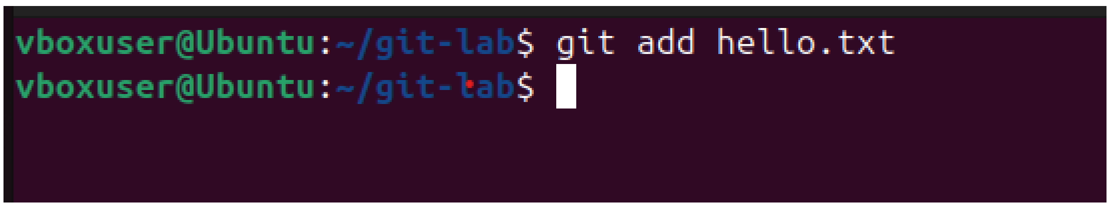
   echo "Hello Git!" > hello.txt



```
vboxuser@Ubuntu:~/git-lab$ echo "Hello Git!" > hello.txt
vboxuser@Ubuntu:~/git-lab$
```

4. **Add the file to the staging area:**

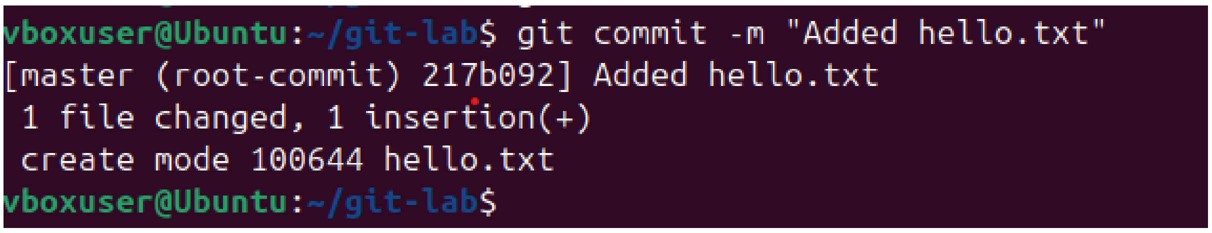
   git add hello.txt



```
vboxuser@Ubuntu:~/git-lab$ git add hello.txt
vboxuser@Ubuntu:~/git-lab$
```

5. **Commit the file with a message:**

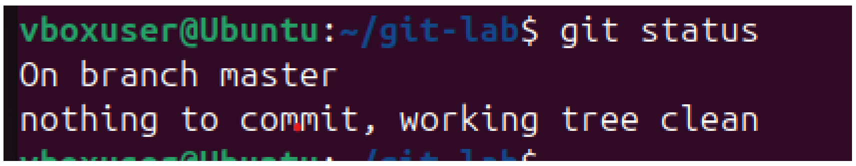
   git commit -m "Added hello.txt"



```
vboxuser@Ubuntu:~/git-lab$ git commit -m "Added hello.txt"
[master (root-commit) 217b092] Added hello.txt
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
vboxuser@Ubuntu:~/git-lab$
```

6. **Check the status of your repository:**

   git status



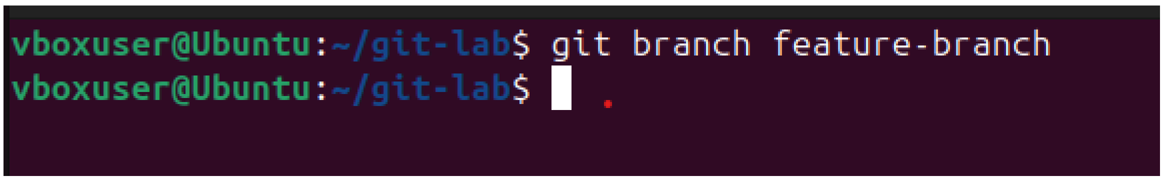
```
vboxuser@Ubuntu:~/git-lab$ git status
On branch master
nothing to commit, working tree clean
```

Lab 4: Creating and Working with Branches

Objective: Understand how to create branches and switch between them to work on different features independently.

Steps:

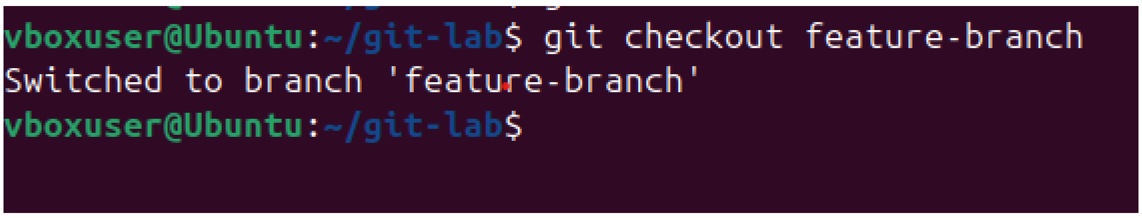
1. **Create a new branch:**
   git branch feature-branch



```
vboxuser@Ubuntu:~/git-lab$ git branch feature-branch
vboxuser@Ubuntu:~/git-lab$
```

2. **Switch to the new branch:**

   git checkout feature-branch



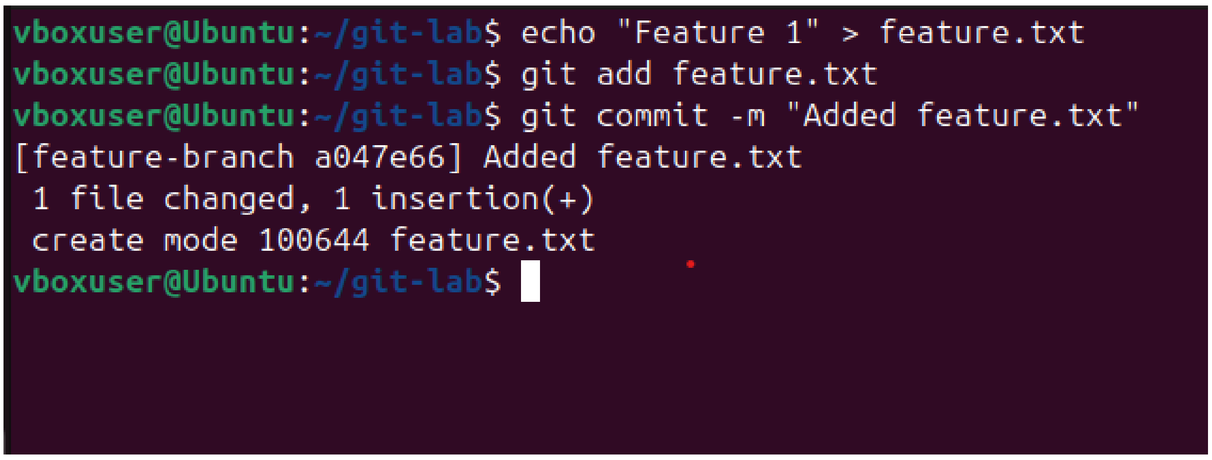
```
vboxuser@Ubuntu:~/git-lab$ git checkout feature-branch
Switched to branch 'feature-branch'
vboxuser@Ubuntu:~/git-lab$
```

3. **Create a new file in the feature branch:**

   echo "Feature 1" > feature.txt

   git add feature.txt

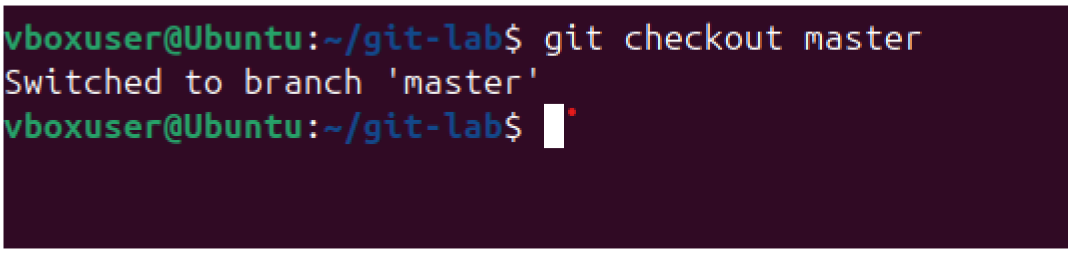
   git commit -m "Added feature.txt"



```
vboxuser@Ubuntu:~/git-lab$ echo "Feature 1" > feature.txt
vboxuser@Ubuntu:~/git-lab$ git add feature.txt
vboxuser@Ubuntu:~/git-lab$ git commit -m "Added feature.txt"
[feature-branch a047e66] Added feature.txt
 1 file changed, 1 insertion(+)
 create mode 100644 feature.txt
vboxuser@Ubuntu:~/git-lab$
```

4. **Switch back to the master branch:**

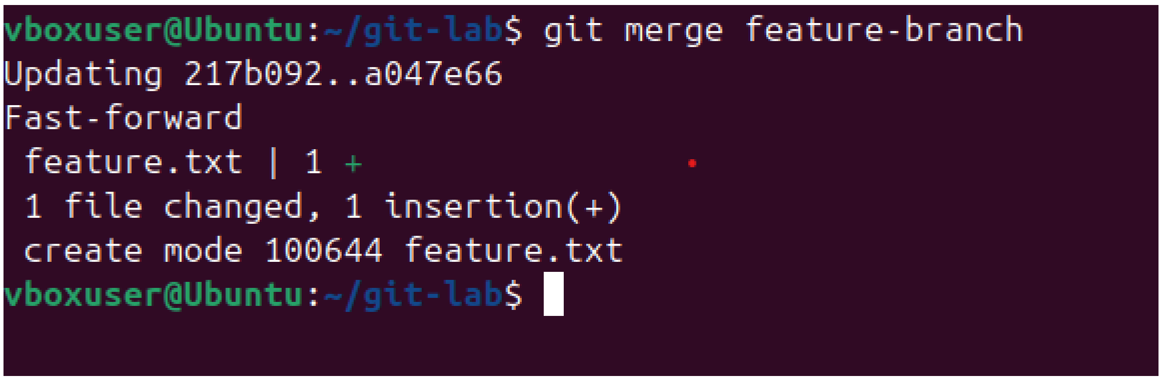
   git checkout master



```
vboxuser@Ubuntu:~/git-lab$ git checkout master
Switched to branch 'master'
vboxuser@Ubuntu:~/git-lab$
```

5. **Merge the changes from feature-branch into master:**

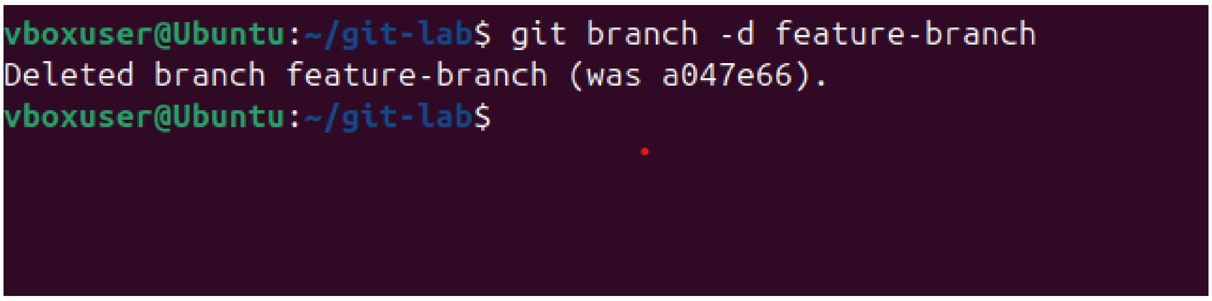
   git merge feature-branch



```
vboxuser@Ubuntu:~/git-lab$ git merge feature-branch
Updating 217b092..a047e66
Fast-forward
 feature.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 feature.txt
vboxuser@Ubuntu:~/git-lab$ 
```

6. **Delete the feature branch (optional):**

   git branch -d feature-branch



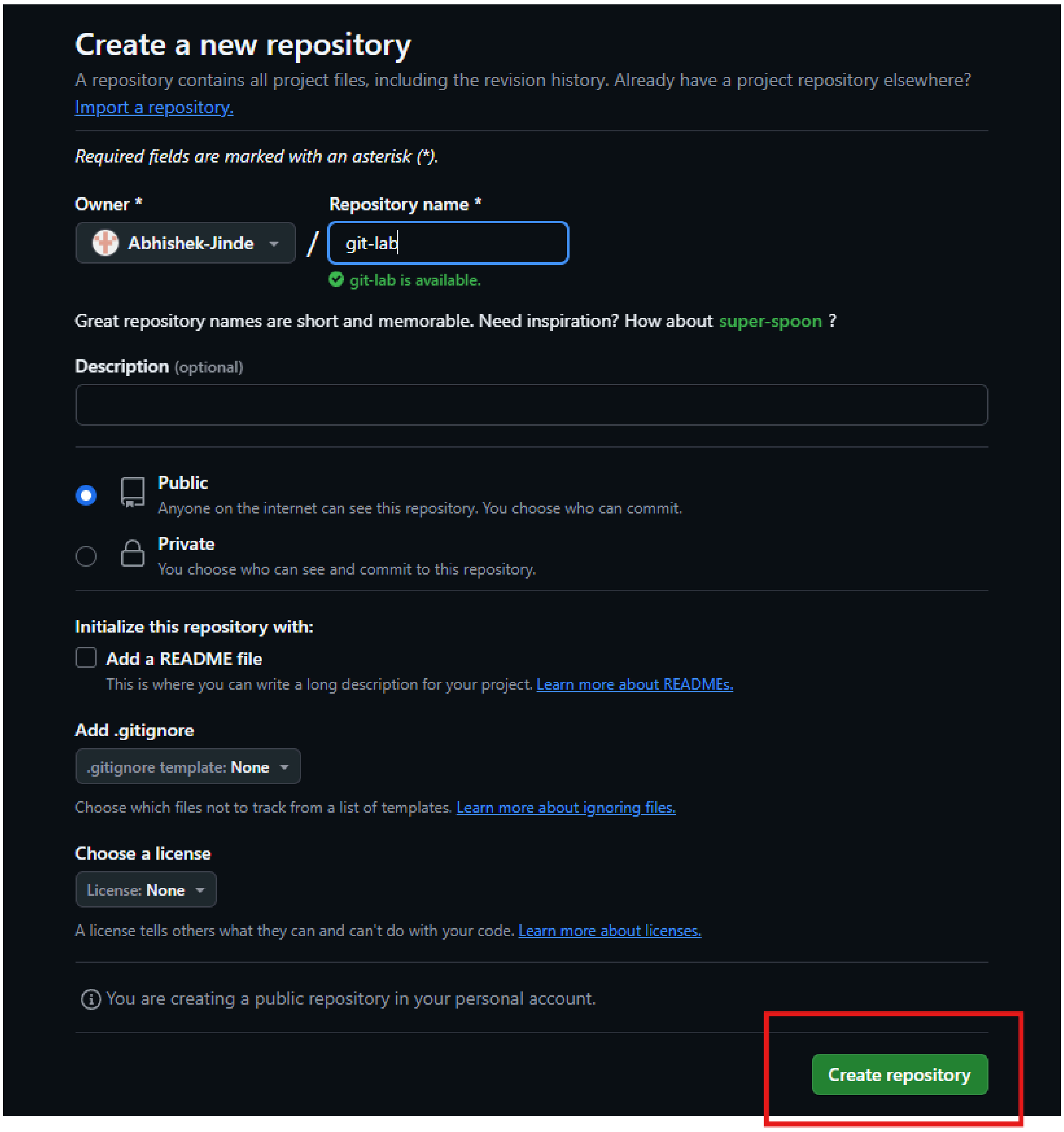
```
vboxuser@Ubuntu:~/git-lab$ git branch -d feature-branch
Deleted branch feature-branch (was a047e66).
vboxuser@Ubuntu:~/git-lab$
```

Lab 5: Working with Remote Repositories on GitHub

Objective: Connect a local Git repository to a GitHub remote repository and push changes.

Steps:

1. **Create a new repository on GitHub (e.g., git-lab).**



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner *: Abhishek-Jinde / Repository name *: git-lab

git-lab is available.

Great repository names are short and memorable. Need inspiration? How about super-spoon ?

Description (optional)

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file — This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore — .gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

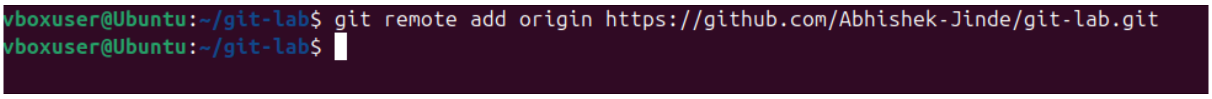
Choose a license — License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

You are creating a public repository in your personal account.

2. Link the local Git repository to the GitHub repository:

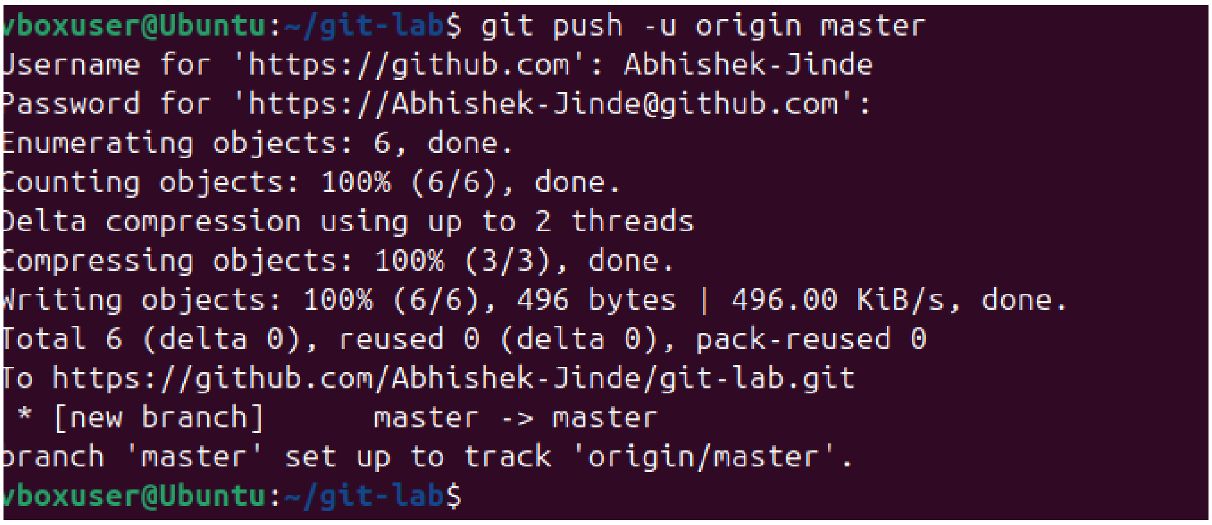
git remote add origin https://github.com/your-username/git-lab.git



```
vboxuser@Ubuntu:~/git-lab$ git remote add origin https://github.com/Abhishek-Jinde/git-lab.git
vboxuser@Ubuntu:~/git-lab$ 
```

3. Push your changes to GitHub:

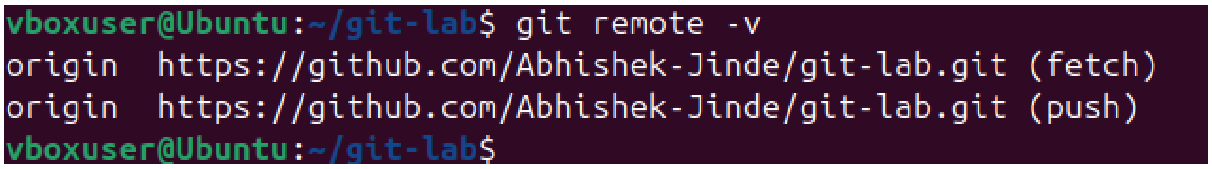
git push -u origin main



```
vboxuser@Ubuntu:~/git-lab$ git push -u origin master
Username for 'https://github.com': Abhishek-Jinde
Password for 'https://Abhishek-Jinde@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 496 bytes | 496.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Abhishek-Jinde/git-lab.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
vboxuser@Ubuntu:~/git-lab$
```

4. To check your remote repositories:

git remote -v



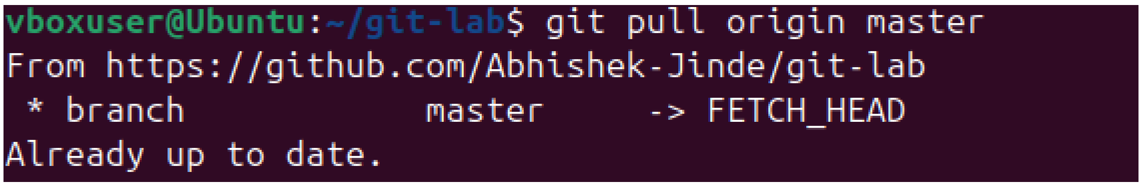
```
vboxuser@Ubuntu:~/git-lab$ git remote -v
origin  https://github.com/Abhishek-Jinde/git-lab.git (fetch)
origin  https://github.com/Abhishek-Jinde/git-lab.git (push)
vboxuser@Ubuntu:~/git-lab$
```

Lab 6: Pulling Changes from GitHub

Objective: Pull changes made in the GitHub repository to your local machine.

Steps:

If changes have been made on GitHub, pull the changes into your local repository:
git pull origin master



```
vboxuser@Ubuntu:~/git-lab$ git pull origin master
From https://github.com/Abhishek-Jinde/git-lab
 * branch            master     -> FETCH_HEAD
Already up to date.
```

Lab 7: Forcing a Merge Conflict

Objective:

Create a scenario where two branches modify the same part of a file, causing a merge conflict when attempting to merge them.

Steps:

1. **Start with a Clean Repository: Ensure you're on the main branch and everything is committed:**

   ```
   git checkout main
   ```

   ```
   git status
   ```

2. **Create and Switch to a Feature Branch: Create a new branch called feature-branch:**

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

3. **Make a Change in feature-branch: Open the hello.txt file and modify line 1:**

   ```
   echo "Feature branch change" > hello.txt
   ```

4. **Add and Commit the Change: Stage and commit the change on the feature-branch:**

   ```
   git add hello.txt
   ```

   ```
   git commit -m "Modified hello.txt in feature-branch"
   ```

5. **Switch Back to the main Branch:**

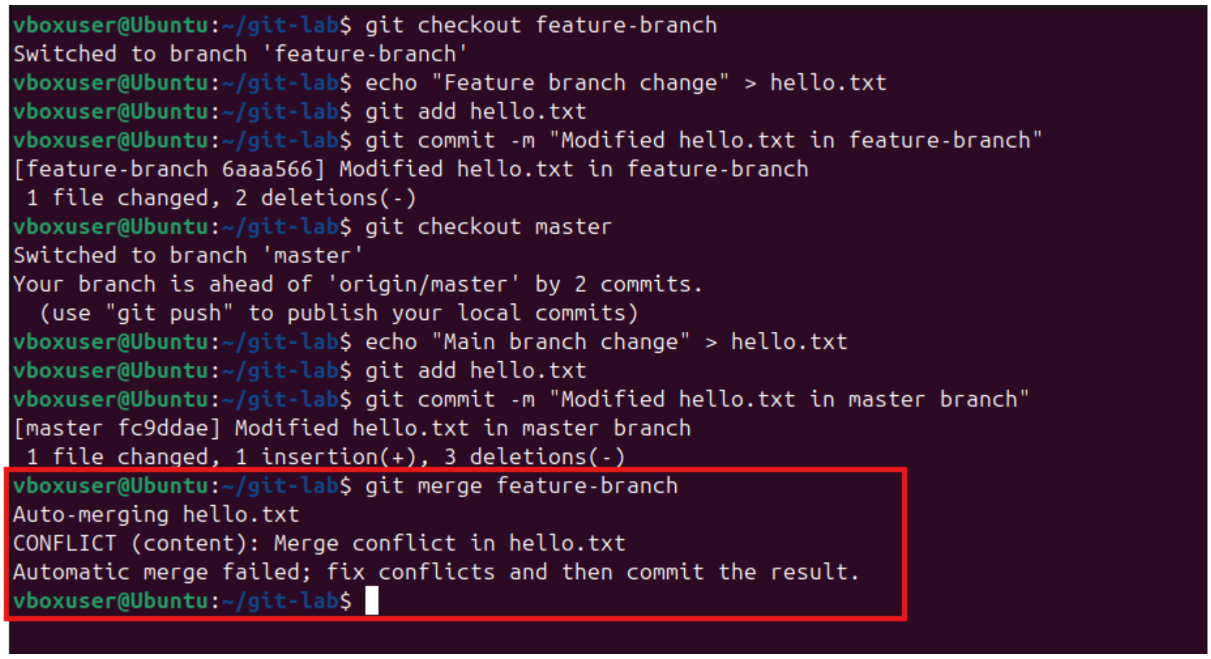
   ```
   git checkout main
   ```

6. **Make a Conflicting Change in main: Modify line 1 of the same hello.txt file:**
   echo "Main branch change" > hello.txt

7. **Add and Commit the Change in main: Stage and commit the change on the main branch:**
   git add hello.txt

   git commit -m "Modified hello.txt in main"

8. **Attempt to Merge feature-branch into main: Now, try to merge feature-branch into main:**
   git merge feature-branch



```
vboxuser@Ubuntu:~/git-lab$ git checkout feature-branch
Switched to branch 'feature-branch'
vboxuser@Ubuntu:~/git-lab$ echo "Feature branch change" > hello.txt
vboxuser@Ubuntu:~/git-lab$ git add hello.txt
vboxuser@Ubuntu:~/git-lab$ git commit -m "Modified hello.txt in feature-branch"
[feature-branch 6aaa566] Modified hello.txt in feature-branch
 1 file changed, 2 deletions(-)
vboxuser@Ubuntu:~/git-lab$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
vboxuser@Ubuntu:~/git-lab$ echo "Main branch change" > hello.txt
vboxuser@Ubuntu:~/git-lab$ git add hello.txt
vboxuser@Ubuntu:~/git-lab$ git commit -m "Modified hello.txt in master branch"
[master fc9ddae] Modified hello.txt in master branch
 1 file changed, 1 insertion(+), 3 deletions(-)
vboxuser@Ubuntu:~/git-lab$ git merge feature-branch
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.
vboxuser@Ubuntu:~/git-lab$
```

9. **Resolving the Conflict:**

   Open hello.txt and you will see both versions of the conflicting changes, like this:

   <<<<<<< HEAD

   Main branch change

   =======

   Feature branch change

>>>>>>> feature-branch

10. **Manually edit the file to resolve the conflict. For example, you can keep both changes:**

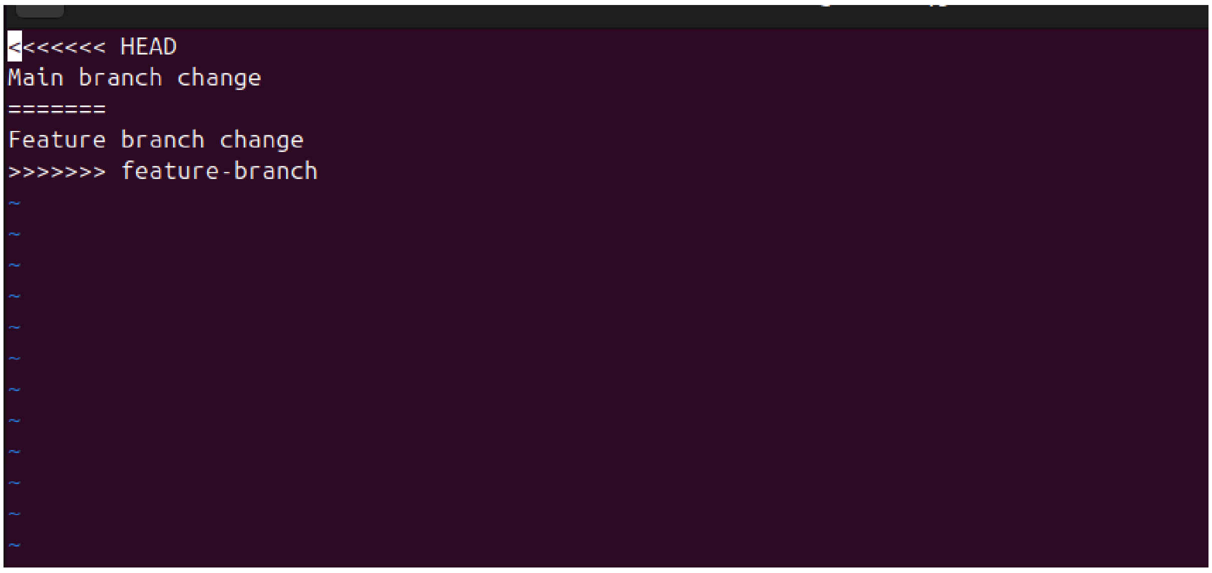
    Main branch change

    Feature branch change

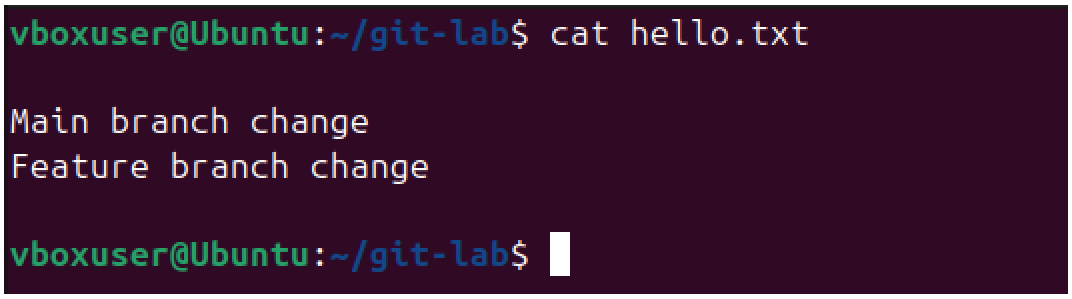
11. **Add and Commit the Resolved File: After resolving the conflict, stage the file and commit the merge:**

    git add hello.txt

    git commit -m "Resolved merge conflict in hello.txt"



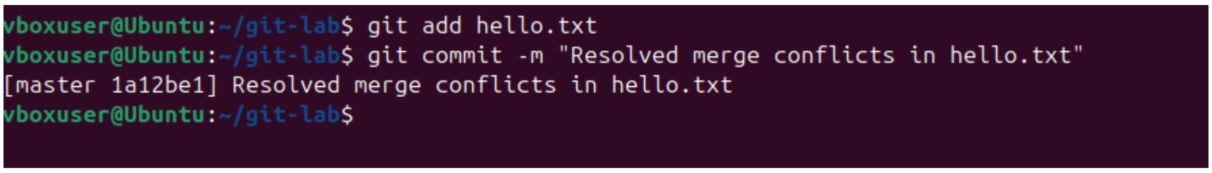
```
<<<<<<< HEAD
Main branch change
=======
Feature branch change
>>>>>>> feature-branch
~
~
~
~
~
~
~
~
~
~
~
~
```



```
vboxuser@Ubuntu:~/git-lab$ cat hello.txt

Main branch change
Feature branch change

vboxuser@Ubuntu:~/git-lab$ 
```



```
vboxuser@Ubuntu:~/git-lab$ git add hello.txt
vboxuser@Ubuntu:~/git-lab$ git commit -m "Resolved merge conflicts in hello.txt"
[master 1a12be1] Resolved merge conflicts in hello.txt
vboxuser@Ubuntu:~/git-lab$
```

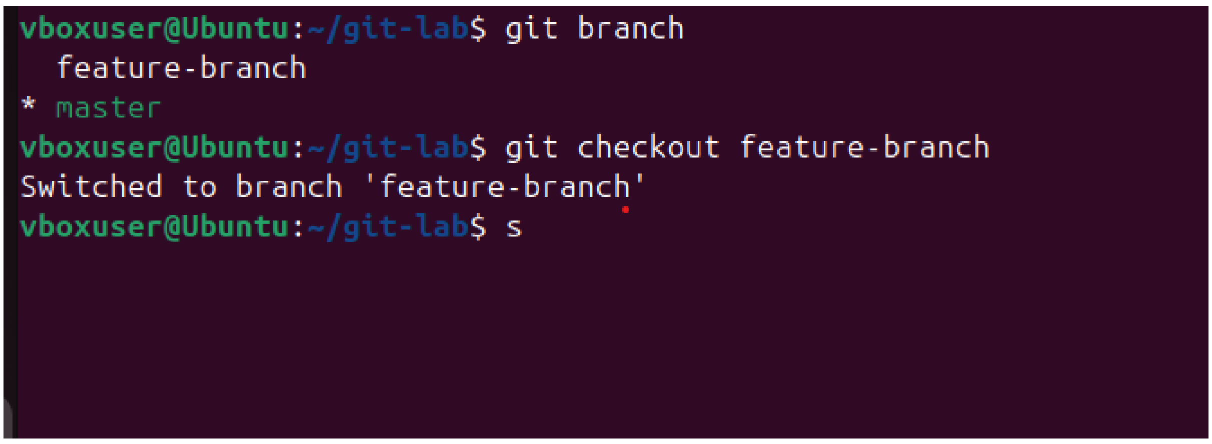
Lab 11: Working with Pull Requests

Steps for Working with Pull Requests

Step 1: Create a New Branch from master

Switch to the master branch to ensure you're starting from the main codebase:
git checkout master

Create a new branch (named feature-branch) and switch to it:
git checkout -b feature-branch



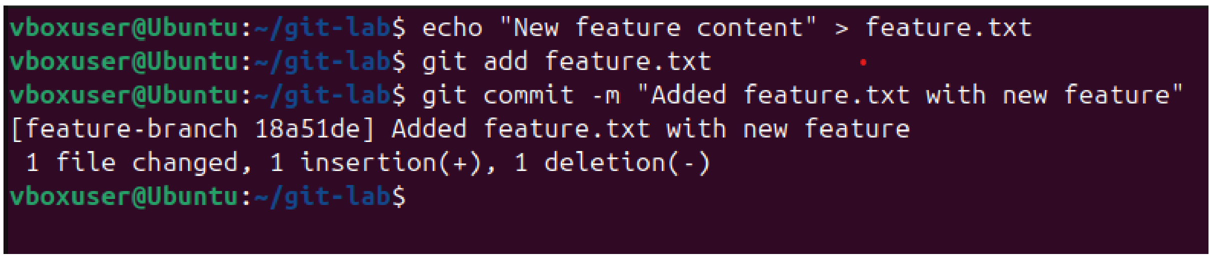
```
vboxuser@Ubuntu:~/git-lab$ git branch
  feature-branch
* master
vboxuser@Ubuntu:~/git-lab$ git checkout feature-branch
Switched to branch 'feature-branch'
vboxuser@Ubuntu:~/git-lab$ s
```

Step 2: Make Changes in feature-branch

Make changes in your feature-branch. For example, add a new file or modify an existing one:
echo "New feature content" > feature.txt

Stage and commit the changes:
git add feature.txt

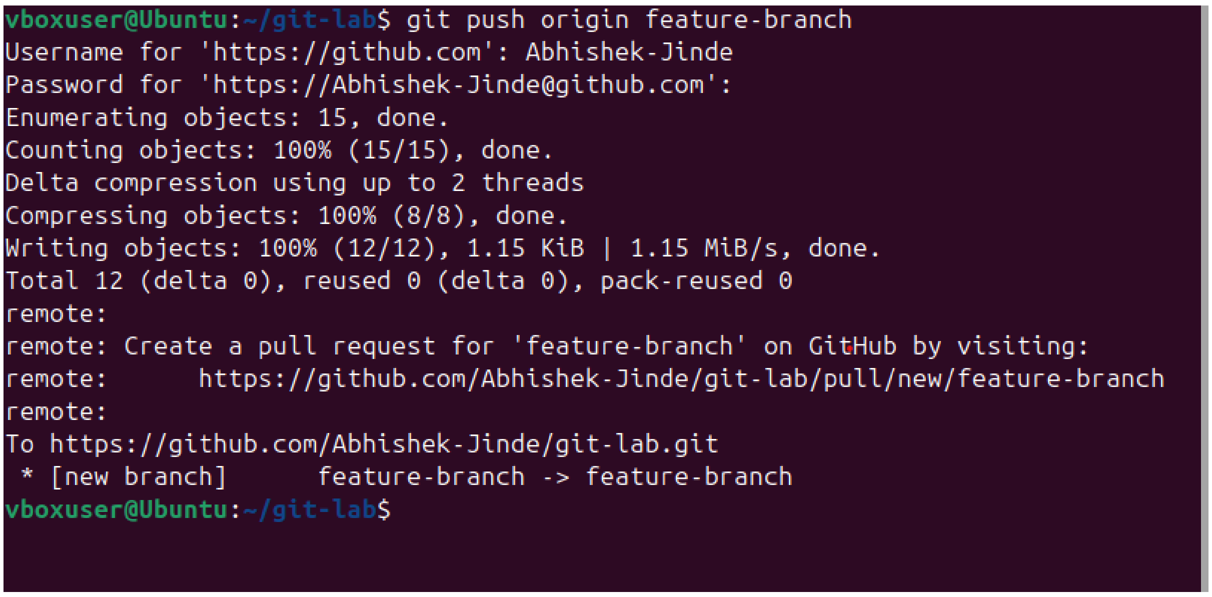
git commit -m "Added feature.txt with new feature"



```
vboxuser@Ubuntu:~/git-lab$ echo "New feature content" > feature.txt
vboxuser@Ubuntu:~/git-lab$ git add feature.txt
vboxuser@Ubuntu:~/git-lab$ git commit -m "Added feature.txt with new feature"
[feature-branch 18a51de] Added feature.txt with new feature
 1 file changed, 1 insertion(+), 1 deletion(-)
vboxuser@Ubuntu:~/git-lab$
```

Step 3: Push feature-branch to GitHub

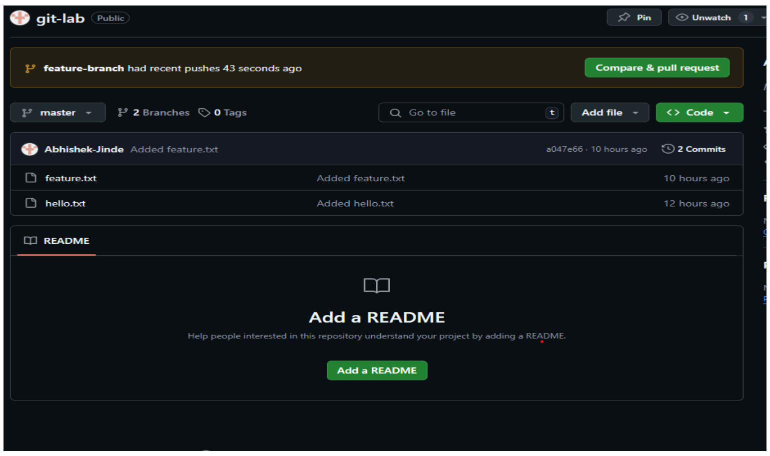
Push the feature-branch to your remote GitHub repository:
git push origin feature-branch



```
vboxuser@Ubuntu:~/git-lab$ git push origin feature-branch
Username for 'https://github.com': Abhishek-Jinde
Password for 'https://Abhishek-Jinde@github.com':
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 2 threads
Compressing objects: 100% (8/8), done.
Writing objects: 100% (12/12), 1.15 KiB | 1.15 MiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'feature-branch' on GitHub by visiting:
remote:      https://github.com/Abhishek-Jinde/git-lab/pull/new/feature-branch
remote:
To https://github.com/Abhishek-Jinde/git-lab.git
 * [new branch]      feature-branch -> feature-branch
vboxuser@Ubuntu:~/git-lab$
```

Step 4: Create a Pull Request (PR) on GitHub

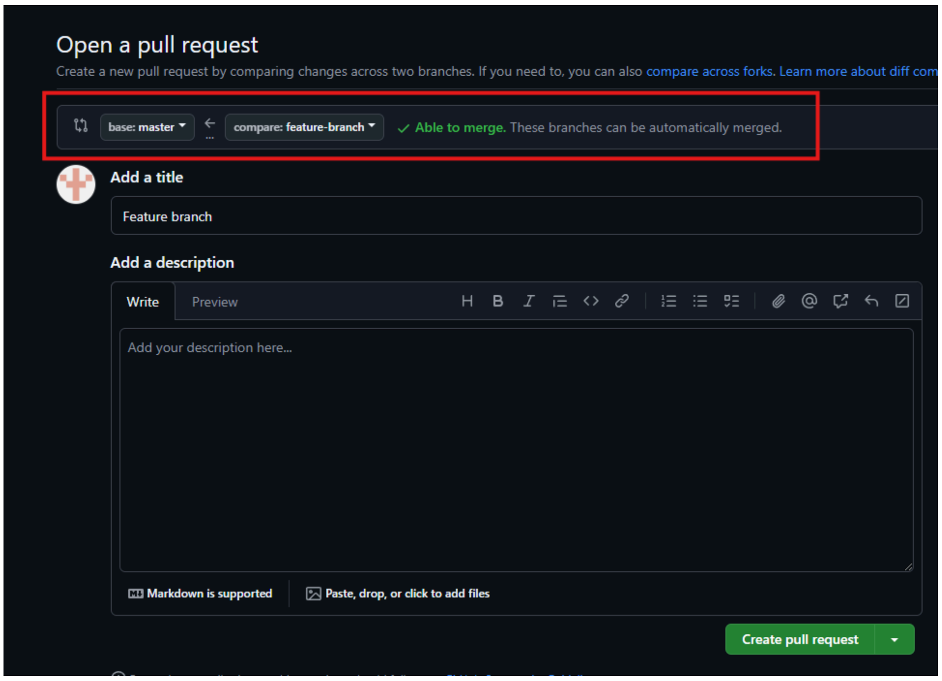
1. Go to your repository on GitHub in a web browser.
2. You will see a prompt to "Compare & pull request" for the newly pushed feature-branch. Click on this button.
3. Set the base branch to master and then compare the branch to feature-branch.
4. Add a title and description for your pull request (describe the changes you made in feature-branch).
5. Click "Create Pull Request" to submit the PR.



feature-branch had recent pushes 43 seconds ago

Abhishek-Jinde Added feature.txt		a047e66 · 10 hours ago · 2 Commits
feature.txt	Added feature.txt	10 hours ago
hello.txt	Added hello.txt	12 hours ago

Add a README
Help people interested in this repository understand your project by adding a README.


Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff com

base: master ← compare: feature-branch ✓ Able to merge. These branches can be automatically merged.

Add a title

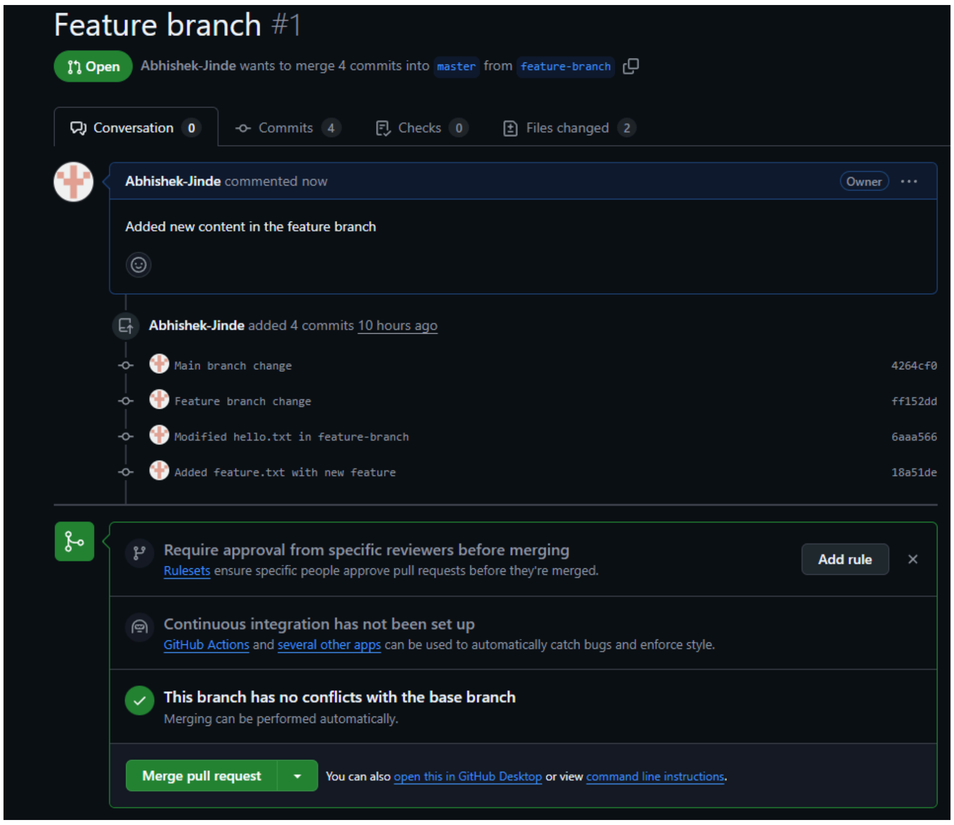
Feature branch

Add a description

Write Preview

Add your description here...

Markdown is supported | Paste, drop, or click to add files


Feature branch #1

Open Abhishek-Jinde wants to merge 4 commits into master from feature-branch

Conversation 0 | Commits 4 | Checks 0 | Files changed 2

Abhishek-Jinde commented now — Owner

Added new content in the feature branch

Abhishek-Jinde added 4 commits 10 hours ago

```
Main branch change                      4264cf0
Feature branch change                   ff152dd
Modified hello.txt in feature-branch    6aaa566
Added feature.txt with new feature      18a51de
```

Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged.

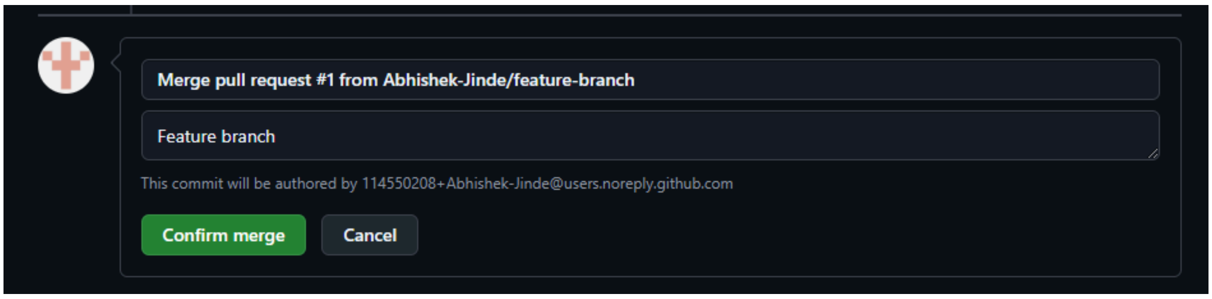
Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

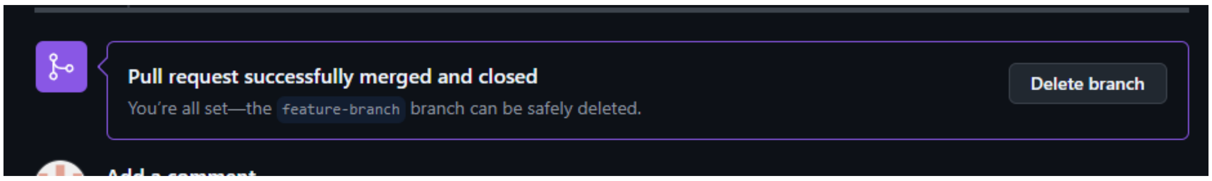
Step 5: Merge the Pull Request

1. Once the PR is approved, you or the repository maintainer can merge the PR.
2. On GitHub, go to the PR page and click "Merge pull request".
3. Confirm the merge by clicking "Confirm merge".


Merge pull request #1 from Abhishek-Jinde/feature-branch

Feature branch

This commit will be authored by 114550208+Abhishek-Jinde@users.noreply.github.com


Pull request successfully merged and closed

You're all set—the `feature-branch` branch can be safely deleted.

1. Git revert vs. Git reset

Scenario:

You've made a commit, but now you want to undo it. Should you use git revert or git reset?

Git revert

- Purpose: git revert creates a new commit that undoes the changes from a previous commit without altering the commit history. This is ideal when you want to maintain the integrity of the history.

Steps:

Make an initial commit:

```
echo "First commit" > file.txt
git add file.txt
git commit -m "Initial commit"
```

Make a second commit:

```
echo "Second commit" >> file.txt
git add file.txt
git commit -m "Second commit"
```

Revert the second commit:

```
git revert HEAD
```

Check the log:
```
git log --oneline
```

Expected Output:

```
abcd123 Revert "Second commit"

efgh456 Second commit

ijkl789 Initial commit
```

Explanation:

- git revert undoes the changes made in the second commit by adding a new commit with the opposite changes. The history remains intact.

Git reset

- Purpose: git reset undoes changes by moving the HEAD pointer back to a previous commit. It can modify the commit history depending on the reset mode.

Steps:
Reset to the first commit:
```
git reset --hard HEAD~1
```

Check the log:
```
git log --oneline
```

Expected Output:

```
ijkl789 Initial commit
```

Explanation:

- git reset --hard moves the HEAD back to the previous commit, erasing the second commit and its changes. This rewrites the commit history.

Key Differences:

- git revert: Keeps the commit history intact by creating a new commit to reverse changes.
- git reset: Removes the commit(s) entirely, which can rewrite the commit history.

---

2. Git revert vs. Git rebase

Scenario:

You've made several commits on a feature branch and want to integrate them into the master branch. You can either use git revert or git rebase.

Git revert

- Purpose: As explained earlier, git revert is used to undo a specific commit. It's not typically used for integrating changes between branches.

Git rebase

- Purpose: git rebase rewrites the commit history by applying the changes from one branch onto another, as if they happened on top of the other branch.

Steps:

Create a feature-branch and make two commits:

```
git checkout -b feature-branch
echo "Feature commit 1" >> file.txt
git add file.txt
git commit -m "Feature commit 1"

echo "Feature commit 2" >> file.txt
git add file.txt
git commit -m "Feature commit 2"
```

Switch back to master:

```
git checkout master
```

Rebase the feature branch onto master:

```
git rebase feature-branch
```

Check the log:

```
git log --oneline
```

Expected Output:

```
mnop123 Feature commit 2
qrst456 Feature commit 1
ijkl789 Initial commit
```

Explanation:

- git rebase takes the commits from feature-branch and re-applies them on top of master, creating a cleaner, linear history.

Key Differences:

- git revert: Reverts a specific commit by adding a new commit.
- git rebase: Re-applies commits from one branch onto another, rewriting history to create a linear sequence.

---

3. Git pull vs. Git fetch

Scenario:

You want to sync your local repository with the remote one. Should you use git pull or git fetch?

Git pull

- Purpose: git pull fetches the latest changes from the remote repository and merges them into your current branch.

Steps:

Make sure you're on master:
git checkout master

Pull changes from the remote repository:
git pull origin master

Expected Output: Git will fetch the changes and immediately merge them into your local master branch.

Git fetch

- Purpose: git fetch only downloads the latest changes from the remote repository without merging them. This allows you to review the changes before applying them.

Steps:
Fetch changes from the remote repository:
git fetch origin

Check the status of the fetched changes:
git log origin/master --oneline

Manually merge the changes (if needed):
git merge origin/master

Explanation:

- git fetch downloads the changes but does not modify your working directory or current branch until you manually merge them.

Key Differences:

- git pull: Fetches and merges changes in one step.
- git fetch: Fetches changes without merging, giving you control over when to apply the updates.

---

4. Git merge

Scenario:

You have a feature-branch and want to merge it into the master branch.

Git merge

- Purpose: git merge combines the changes from one branch into another.

Steps:
Make sure you are on the master branch:

```
git checkout master
```

Merge feature-branch into master:

```
git merge feature-branch
```

Check the log:

```
git log --oneline
```

Expected Output:

```
mnop123 Merge branch 'feature-branch'
qrst456 Feature commit 2
ijkl789 Feature commit 1
abcd123 Initial commit
```

Explanation:

- git merge incorporates changes from the feature-branch into the master branch. If there are no conflicts, Git will automatically complete the merge.

Conclusion

- git revert vs git reset: Use revert to keep history intact, and reset to remove commits entirely.
- git revert vs git rebase: revert is for undoing specific commits, while rebase is for rewriting history by reapplying commits.
- git pull vs git fetch: pull fetches and merges changes, while fetch only downloads changes without applying them.
- git merge: Used to combine changes from one branch into another.