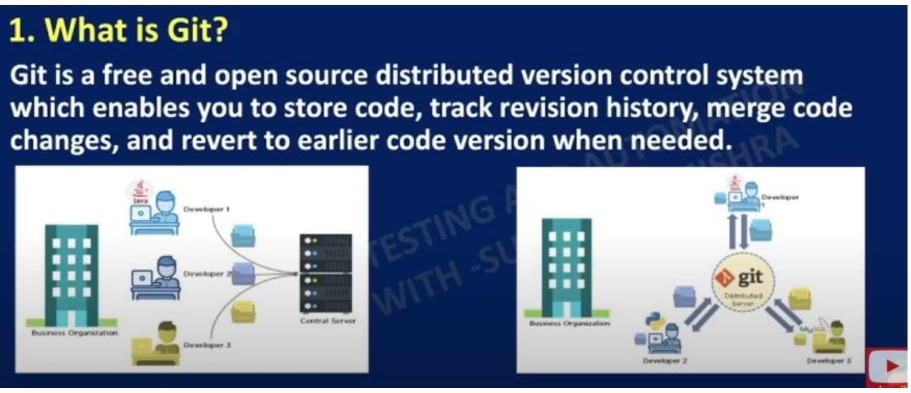
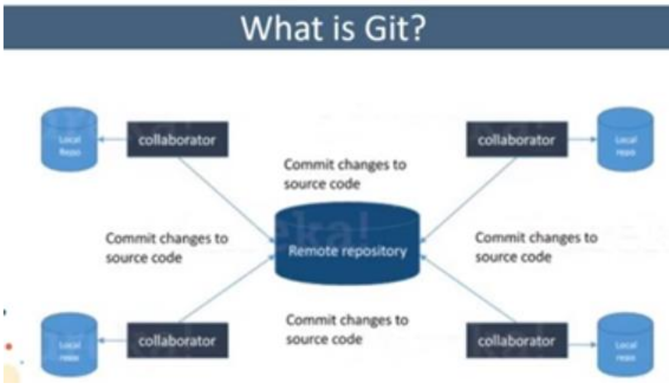
GIT



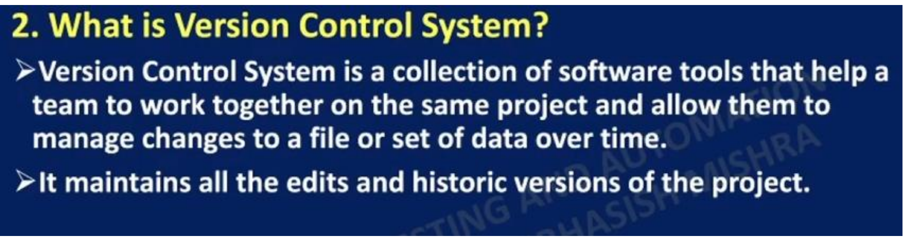
1. What is Git?

Git is a free and open source distributed version control system which enables you to store code, track revision history, merge code changes, and revert to earlier code version when needed.



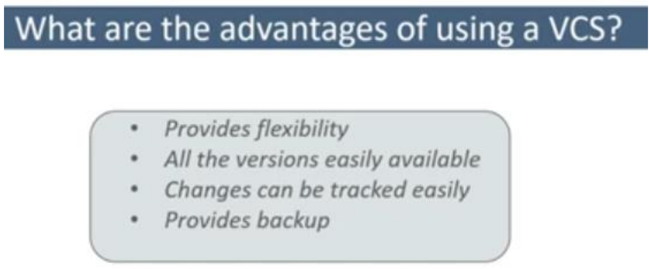
What is Git?

collaborator
collaborator
collaborator
collaborator
Local Repo
Local repo
Local repo
Local repo
Commit changes to source code
Commit changes to source code
Commit changes to source code
Commit changes to source code
Remote repository



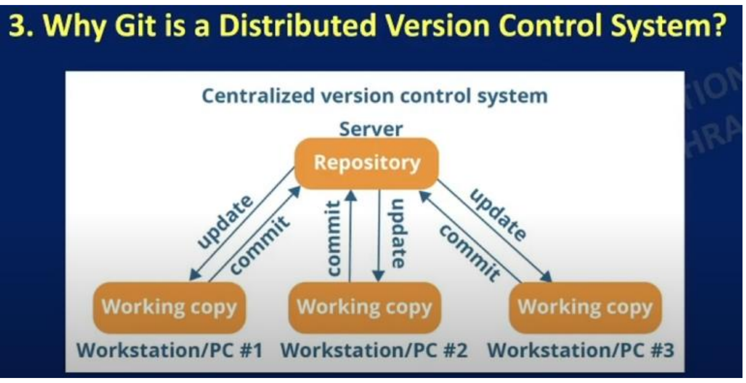
2. What is Version Control System?

- ➢Version Control System is a collection of software tools that help a team to work together on the same project and allow them to manage changes to a file or set of data over time.
- ➢It maintains all the edits and historic versions of the project.

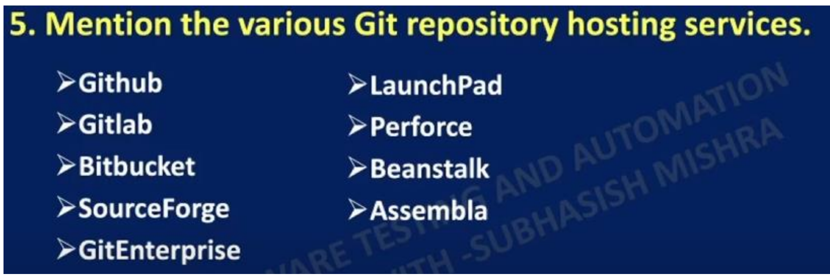


What are the advantages of using a VCS?

- *Provides flexibility*
- *All the versions easily available*
- *Changes can be tracked easily*
- *Provides backup*

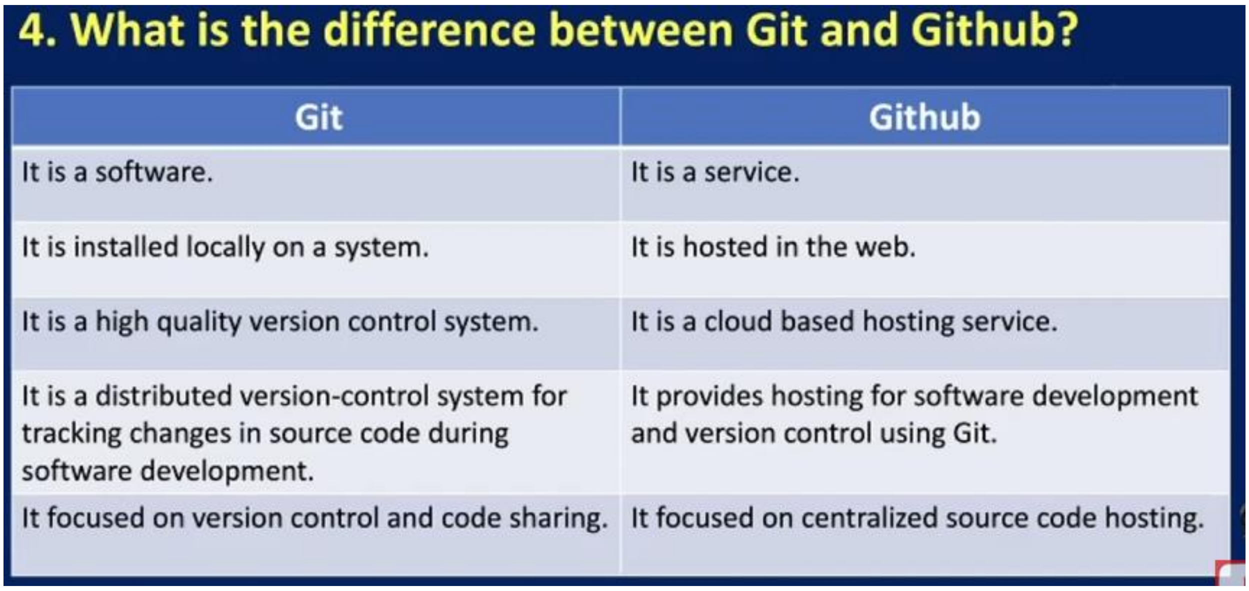


3. Why Git is a Distributed Version Control System?



5. Mention the various Git repository hosting services.

➢Github	➢LaunchPad
➢Gitlab	➢Perforce
➢Bitbucket	➢Beanstalk
➢SourceForge	➢Assembla
➢GitEnterprise	



4. What is the difference between Git and Github?

Git	Github
It is a software.	It is a service.
It is installed locally on a system.	It is hosted in the web.
It is a high quality version control system.	It is a cloud based hosting service.
It is a distributed version-control system for tracking changes in source code during software development.	It provides hosting for software development and version control using Git.
It focused on version control and code sharing.	It focused on centralized source code hosting.

6. What is a Git repository?

- Git repository is a place where all the Git files are stored.
- These files can either be stored on the local repository or on the remote repository.
- It allows us to save versions of our code which we can access whenever needed.

7. How can you initialize a repository in Git?

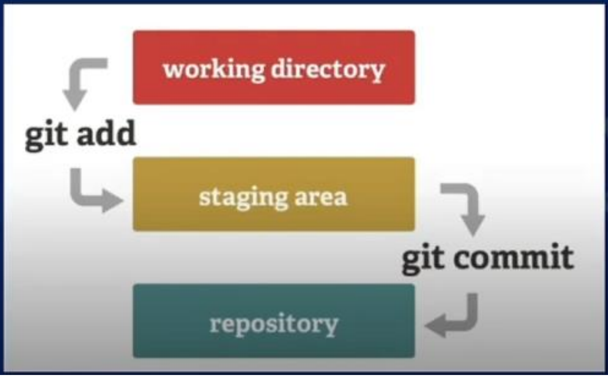
Using git init command

8. What are the states of a file in Git?

Three different states.

- Modified
- Staged
- Committed

9. What Is Staging Area In GIT?



10. Name a few Git commands with their usage.

git init: To initialize an empty Git repository

git config: To configure a username and email address

git add: To add one or more files to the staging area

git diff: To view the changes made to a file

git commit: To commit changes to head but not to the remote repository.

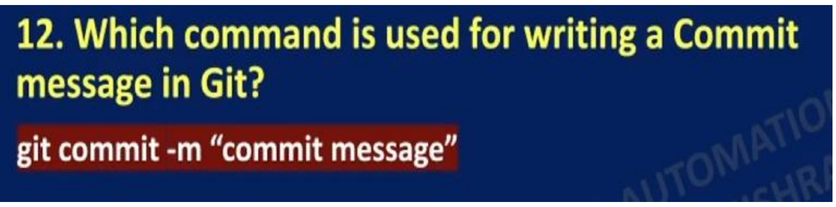
git log :- list the version history for the current branch.

git checkout [branch name] :- used to switch from one branch to another.

11. What are the advantages of using Git?

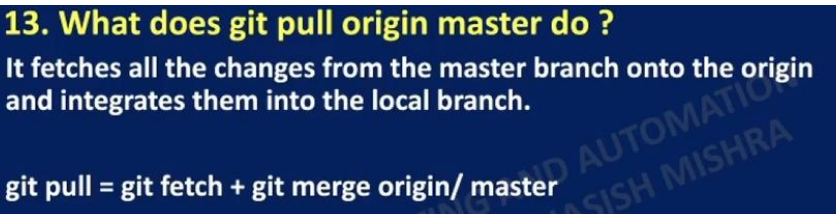
- Distributed manner of development and easy team collaboration
- Widespread acceptance
- Maintains the integrity of source code
- Branching Capabilities
- Faster release cycles

SOFTWARE TESTING AND AUTOMATION LEARN WITH -SUBHASISH MISHRA



12. Which command is used for writing a Commit message in Git?

git commit -m "commit message"

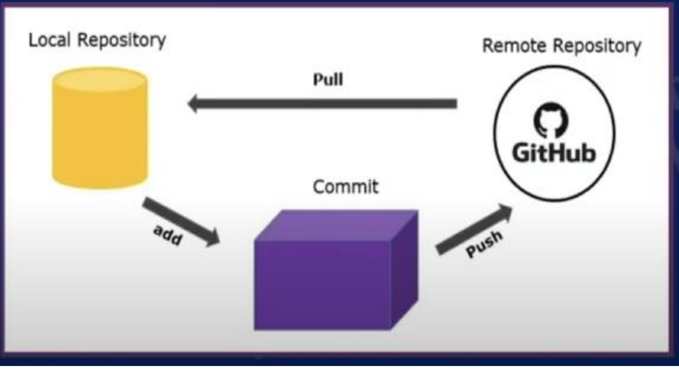


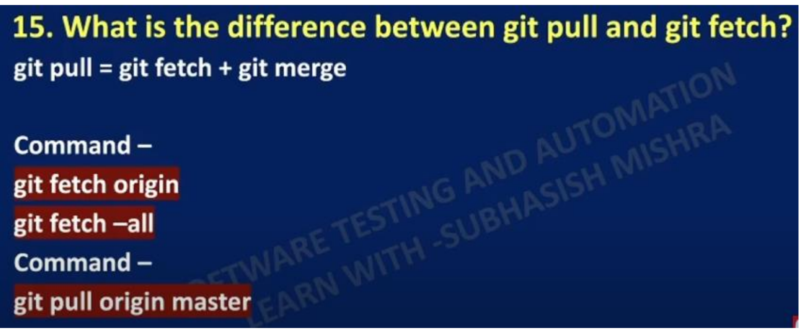
13. What does git pull origin master do ?

It fetches all the changes from the master branch onto the origin and integrates them into the local branch.

git pull = git fetch + git merge origin/ master

14. What does the git push command do?





15. What is the difference between git pull and git fetch?

git pull = git fetch + git merge

Command –

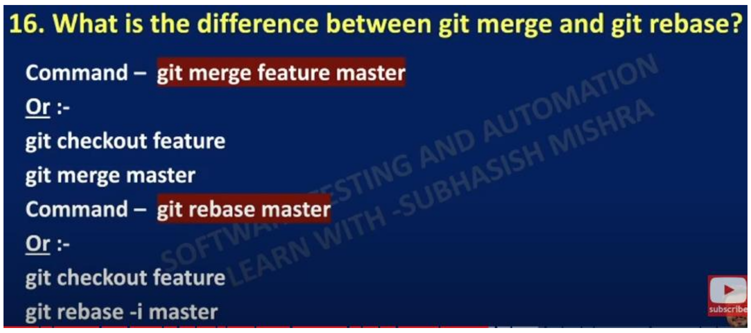
git fetch origin

git fetch –all

Command –

git pull origin master

SOFTWARE TESTING AND AUTOMATION LEARN WITH -SUBHASISH MISHRA



16. What is the difference between git merge and git rebase?

Command – git merge feature master

Or :-

git checkout feature

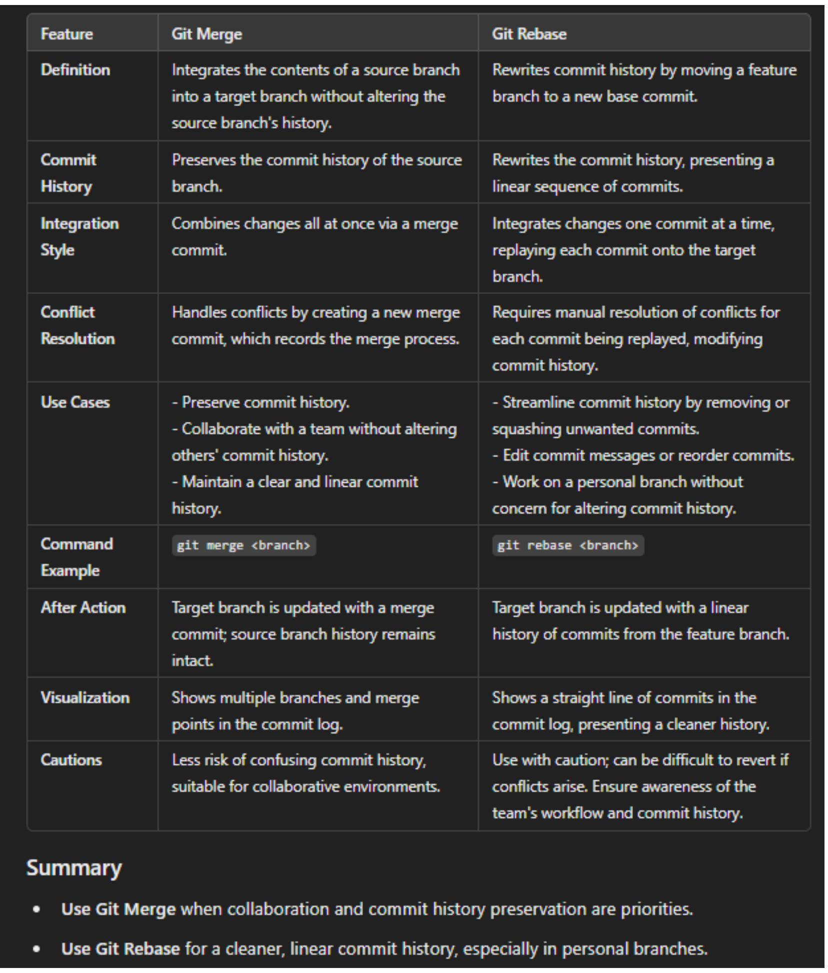
git merge master

Command – git rebase master

Or :-

git checkout feature

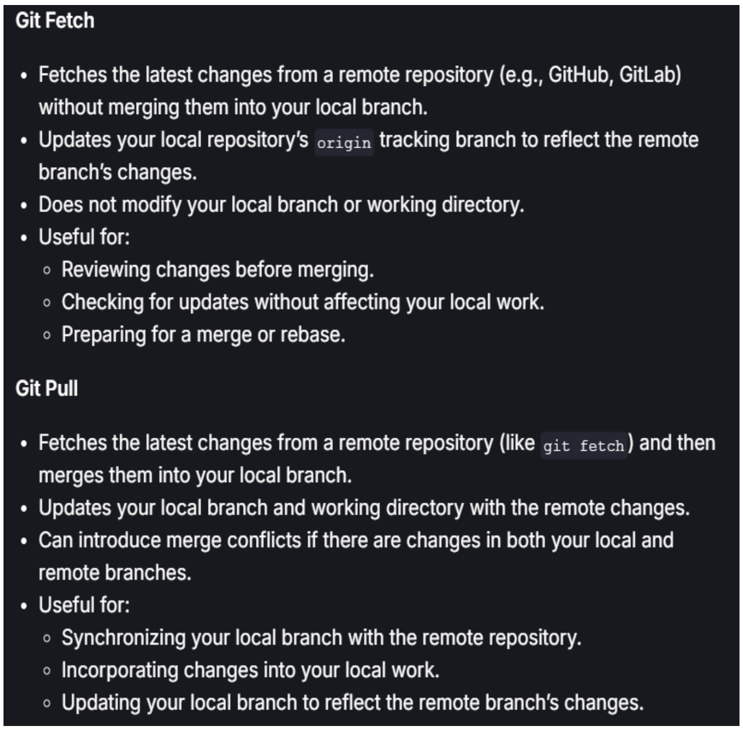
git rebase -i master



Feature	Git Merge	Git Rebase
Definition	Integrates the contents of a source branch into a target branch without altering the source branch's history.	Rewrites commit history by moving a feature branch to a new base commit.
Commit History	Preserves the commit history of the source branch.	Rewrites the commit history, presenting a linear sequence of commits.
Integration Style	Combines changes all at once via a merge commit.	Integrates changes one commit at a time, replaying each commit onto the target branch.
Conflict Resolution	Handles conflicts by creating a new merge commit, which records the merge process.	Requires manual resolution of conflicts for each commit being replayed, modifying commit history.
Use Cases	- Preserve commit history. - Collaborate with a team without altering others' commit history. - Maintain a clear and linear commit history.	- Streamline commit history by removing or squashing unwanted commits. - Edit commit messages or reorder commits. - Work on a personal branch without concern for altering commit history.
Command Example	`git merge <branch>`	`git rebase <branch>`
After Action	Target branch is updated with a merge commit; source branch history remains intact.	Target branch is updated with a linear history of commits from the feature branch.
Visualization	Shows multiple branches and merge points in the commit log.	Shows a straight line of commits in the commit log, presenting a cleaner history.
Cautions	Less risk of confusing commit history, suitable for collaborative environments.	Use with caution; can be difficult to revert if conflicts arise. Ensure awareness of the team's workflow and commit history.

Summary

- **Use Git Merge** when collaboration and commit history preservation are priorities.
- **Use Git Rebase** for a cleaner, linear commit history, especially in personal branches.

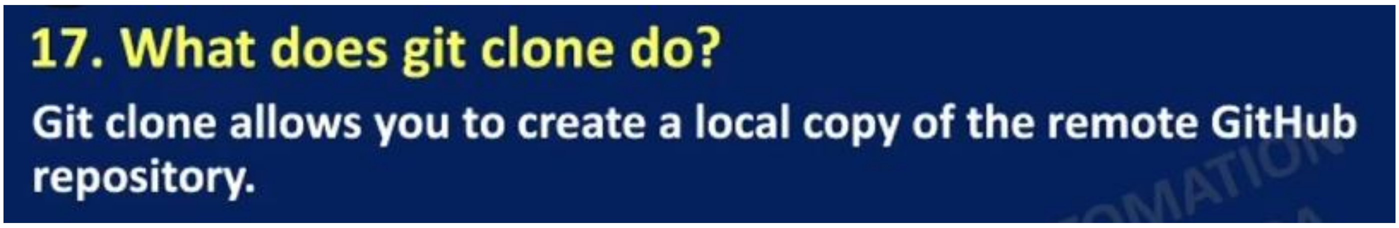


Git Fetch

- Fetches the latest changes from a remote repository (e.g., GitHub, GitLab) without merging them into your local branch.
- Updates your local repository's `origin` tracking branch to reflect the remote branch's changes.
- Does not modify your local branch or working directory.
- Useful for:
  - Reviewing changes before merging.
  - Checking for updates without affecting your local work.
  - Preparing for a merge or rebase.

Git Pull

- Fetches the latest changes from a remote repository (like `git fetch`) and then merges them into your local branch.
- Updates your local branch and working directory with the remote changes.
- Can introduce merge conflicts if there are changes in both your local and remote branches.
- Useful for:
  - Synchronizing your local branch with the remote repository.
  - Incorporating changes into your local work.
  - Updating your local branch to reflect the remote branch's changes.



17. What does git clone do?

Git clone allows you to create a local copy of the remote GitHub repository.

18. What is Git stash ?

GIT stash captures the present state of the working directory and index it and keeps it on the stack at a later stage.

It returns a clean working directory.

19. What does the git reset --mixed and git merge --abort commands do?

`git reset --mixed` is used to undo changes made in the working directory and staging area.

`git merge --abort` is used to stop the merge process and return back to the state before the merging began.

20. How do you find a list of files that has been changed in a particular commit?

```
git diff-tree –r {commit hash}
```

21. What is the functionality of git clean command?

The git clean command removes the untracked files from the working directory.

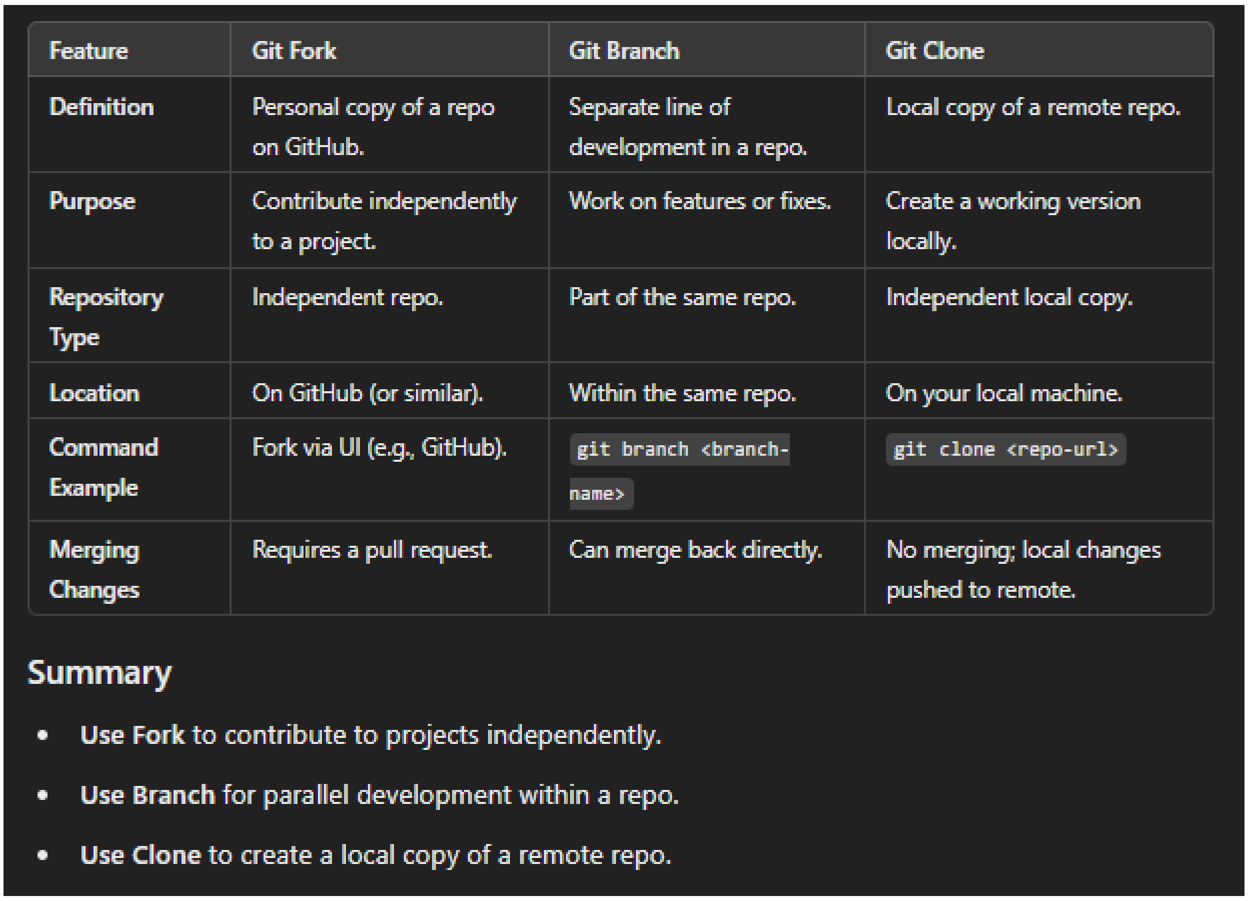
22. What is the difference between fork, branch, and clone?

A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository.

git cloning means pointing to an existing repository and make a copy of that repository in a new directory, at some other location.

In Git, a branch is a new/separate version of the main repository.

SOFTWARE TESTING AND AUTOMATION LEARN WITH SUBHASISH MISHRA

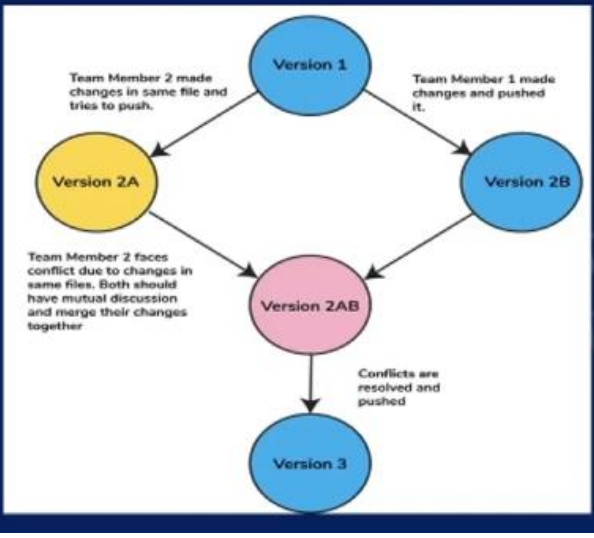


Feature	Git Fork	Git Branch	Git Clone
Definition	Personal copy of a repo on GitHub.	Separate line of development in a repo.	Local copy of a remote repo.
Purpose	Contribute independently to a project.	Work on features or fixes.	Create a working version locally.
Repository Type	Independent repo.	Part of the same repo.	Independent local copy.
Location	On GitHub (or similar).	Within the same repo.	On your local machine.
Command Example	Fork via UI (e.g., GitHub).	`git branch <branch-name>`	`git clone <repo-url>`
Merging Changes	Requires a pull request.	Can merge back directly.	No merging; local changes pushed to remote.

Summary

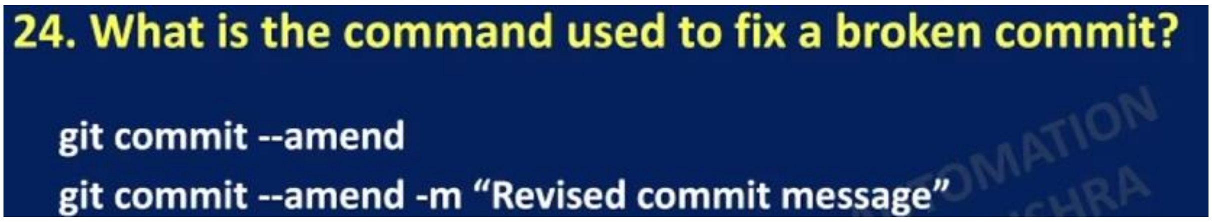
- **Use Fork** to contribute to projects independently.
- **Use Branch** for parallel development within a repo.
- **Use Clone** to create a local copy of a remote repo.

23. How do you resolve conflicts in Git?



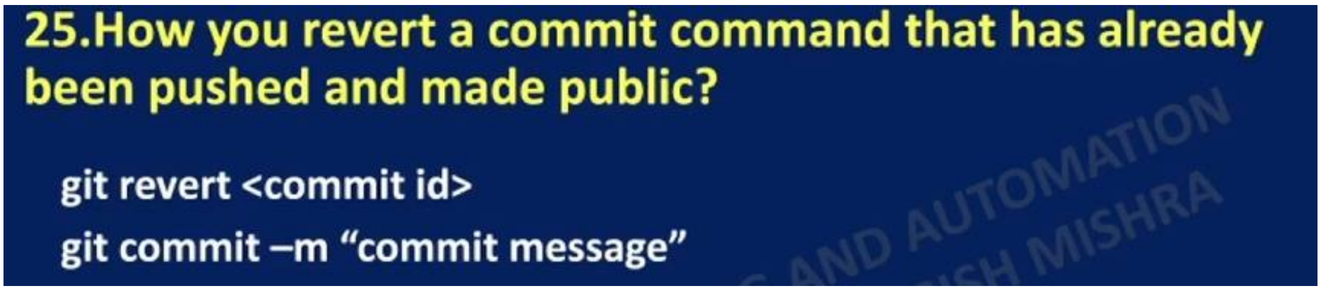
Version 1
Team Member 2 made changes in same file and tries to push.
Team Member 1 made changes and pushed it.
Version 2A
Version 2B
Team Member 2 faces conflict due to changes in same files. Both should have mutual discussion and merge their changes together
Version 2AB
Conflicts are resolved and pushed
Version 3

- ➢Identify the files responsible for the conflicts.
- ➢Implement the desired changes to the files
- ➢Add the files using the git add command.
- ➢The last step is to commit the changes in the file with the help of the git commit command.



24. What is the command used to fix a broken commit?

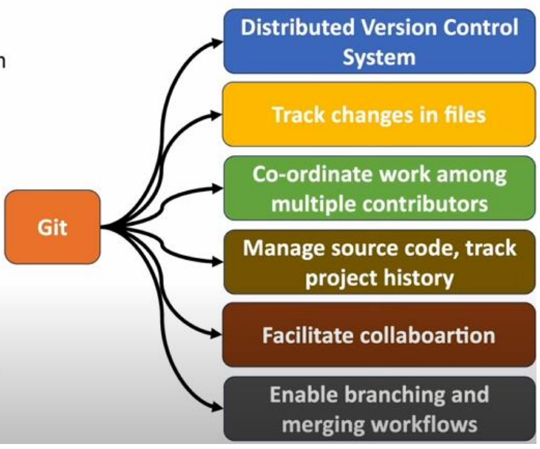
```
git commit --amend
git commit --amend -m "Revised commit message"
```



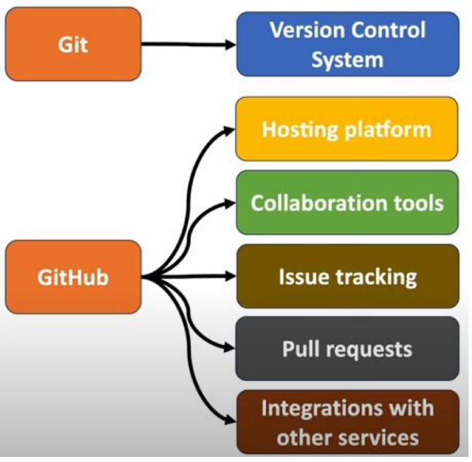
25.How you revert a commit command that has already been pushed and made public?

```
git revert <commit id>
git commit –m "commit message"
```

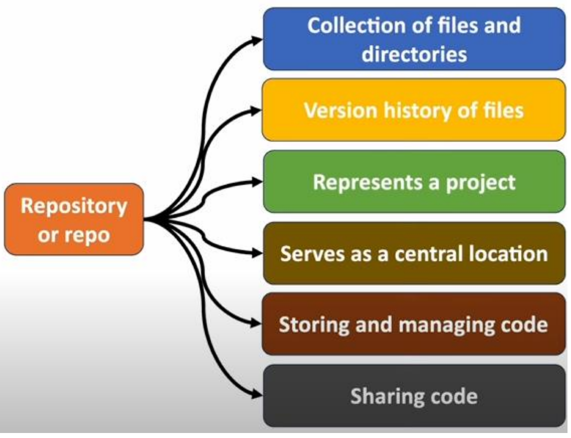
- Git is a distributed version control system designed to track changes in files and coordinate work among multiple contributors.
- It is used to manage source code, track project history, facilitate collaboration, and enable branching and merging workflows.


Git
Distributed Version Control System
Track changes in files
Co-ordinate work among multiple contributors
Manage source code, track project history
Facilitate collaboartion
Enable branching and merging workflows

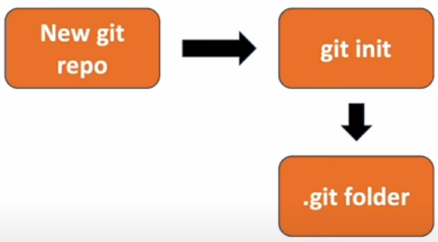
- Git is the version control system itself, while GitHub is a hosting platform for Git repositories.
- GitHub provides additional features such as collaboration tools, issue tracking, pull requests, and integrations with other services.



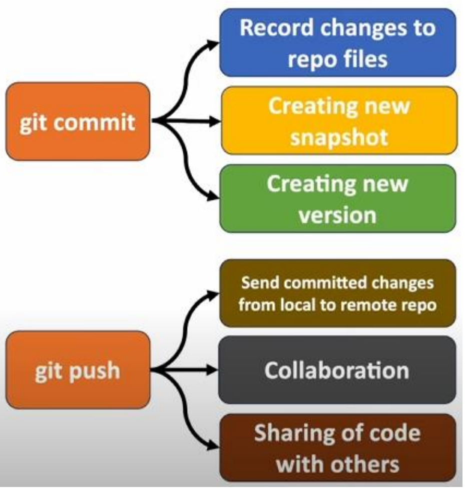
- A repository, or repo, in Git is a collection of files and directories along with the version history of those files.
- It represents a project and serves as the central location for storing, managing, and sharing code.


Repository or repo
Collection of files and directories
Version history of files
Represents a project
Serves as a central location
Storing and managing code
Sharing code

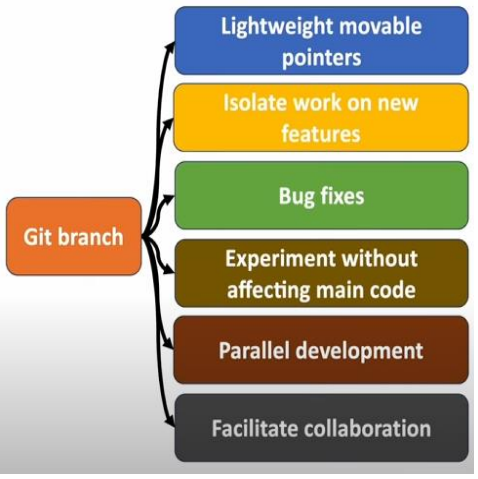
- To create a new Git repository, navigate to the project directory and run the command **git init.**
- This initializes a new Git repository in the current directory, creating a hidden **.git** folder to store version control metadata.



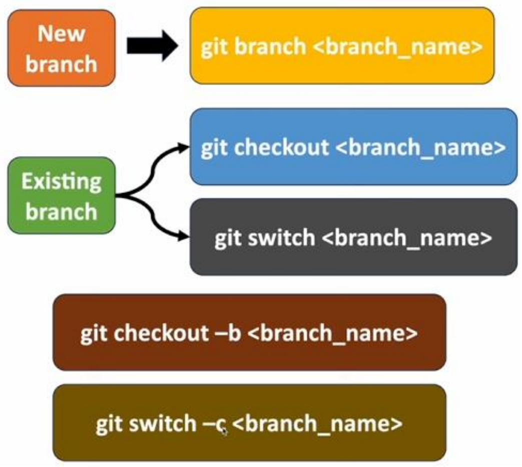
- A **git commit** records changes to the repository's files, creating a new snapshot or version of the project.
- A **git push**, on the other hand, sends committed changes from a local repository to a remote repository, enabling collaboration and sharing of code with others.


git commit
Record changes to repo files
Creating new snapshot
Creating new version
git push
Send committed changes from local to remote repo
Collaboration
Sharing of code with others

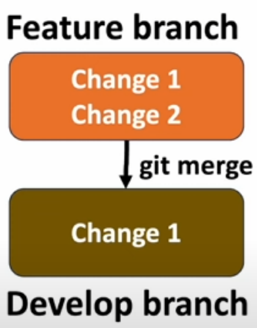
- A **Git branch** is a lightweight movable pointer to a commit in the repository's version history.
- Branches are used to isolate work on new features, bug fixes, or experiments without affecting the main codebase.
- They enable parallel development and facilitate collaboration among team members.


Git branch
Lightweight movable pointers
Isolate work on new features
Bug fixes
Experiment without affecting main code
Parallel development
Facilitate collaboration

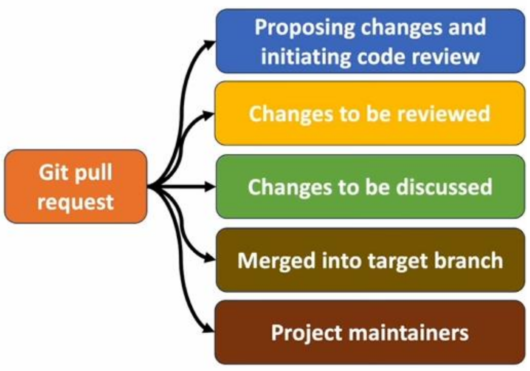
- To create a new branch, use the command **git branch <branch_name>.**
- To switch to an existing branch, use **git checkout <branch_name>** or **git switch <branch_name>**.
- Alternatively, you can create and switch to a new branch in one step using **git checkout -b <branch_name>** or **git switch -c <branch_name>**.


New branch
git branch <branch_name>
Existing branch
git checkout <branch_name>
git switch <branch_name>
git checkout –b <branch_name>
git switch –c <branch_name>

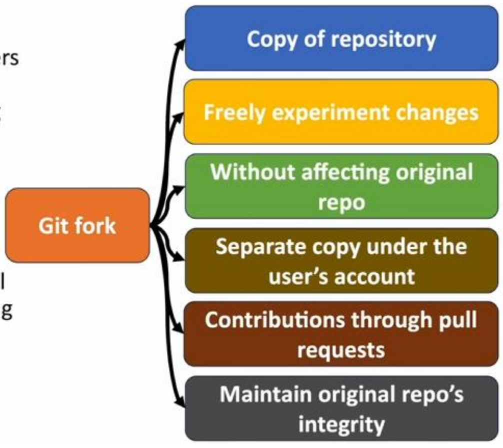
- A **Git merge** combines changes from one branch into another branch.
- It integrates the changes made in the source branch (e.g., feature branch) into the target branch (e.g., main branch) by creating a new merge commit that incorporates both sets of changes.



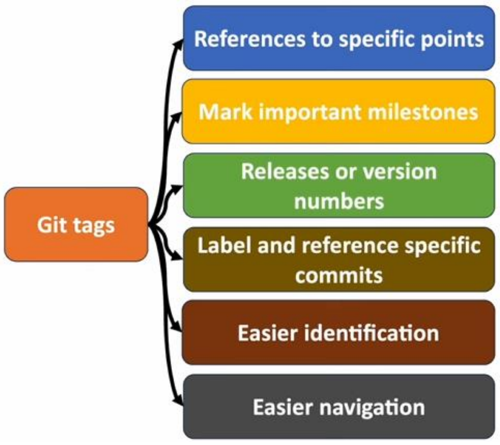
- A **Git pull request** is a mechanism for proposing changes and initiating code review in a collaborative development workflow.
- It allows contributors to request that their changes be reviewed, discussed, and eventually merged into the target branch by project maintainers.



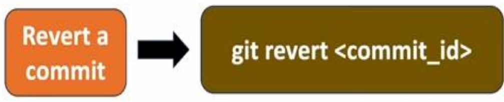
- A **Git fork** is a copy of a repository that allows users to freely experiment with changes without affecting the original repository.
- In GitHub, forking a repository creates a separate copy under the user's account, enabling contributions through pull requests while maintaining the original repository's integrity.


Git fork
Copy of repository
Freely experiment changes
Without affecting original repo
Separate copy under the user's account
Contributions through pull requests
Maintain original repo's integrity

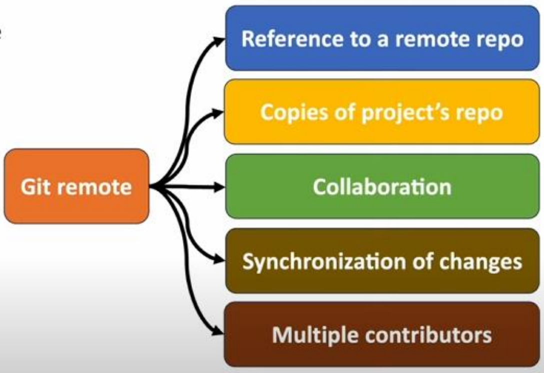
- **Git tags** are references to specific points in a repository's version history, typically used to mark important milestones such as releases or version numbers.
- They provide a way to label and reference specific commits for easier identification and navigation.


Git tags
References to specific points
Mark important milestones
Releases or version numbers
Label and reference specific commits
Easier identification
Easier navigation

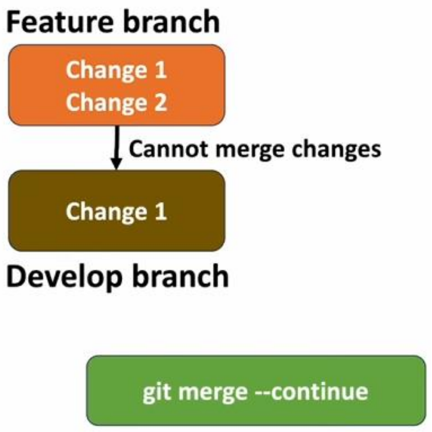
- To revert a commit in Git, use the command **git revert <commit_id>**.
- This creates a new commit that undoes the changes introduced by the specified commit, effectively reverting the repository to its previous state without rewriting history.



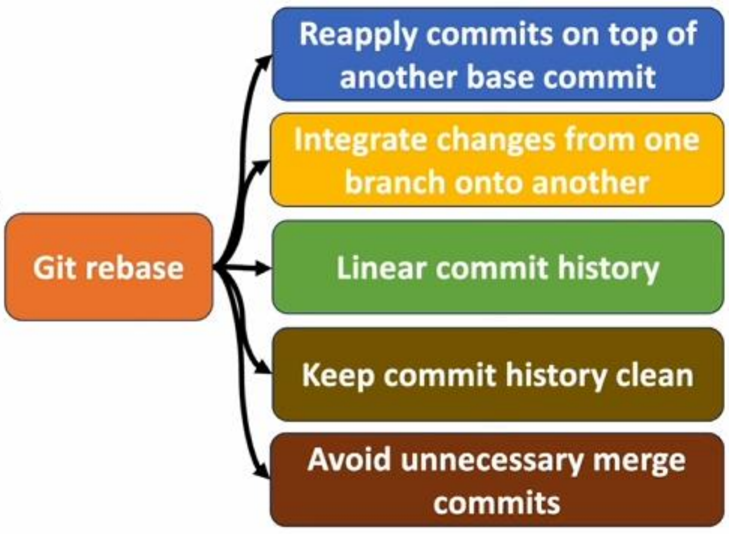
- **Git remote** is a reference to a remote repository, typically hosted on a server such as GitHub, GitLab, or Bitbucket.
- Remote repositories are copies of a project's repository stored on remote servers, allowing collaboration and synchronization of changes between multiple contributors.



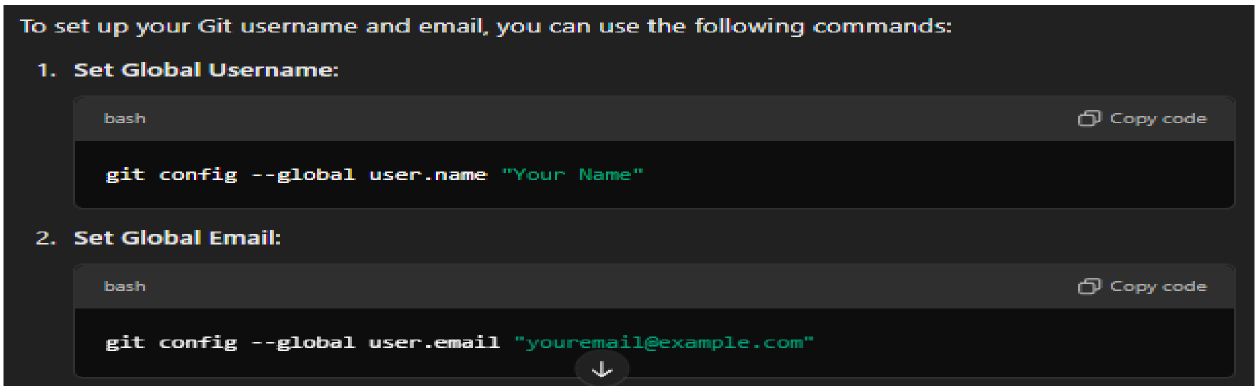
- Merge conflicts occur when Git cannot automatically merge changes from different branches.
- To resolve conflicts, manually edit the conflicting files to resolve conflicts, mark them as resolved using git add, and commit the changes.
- Finally, complete the merge process using **git merge --continue**.



- **Git rebase** is a command used to reapply commits on top of another base commit.
- It is commonly used to integrate changes from one branch onto another while maintaining a linear commit history.
- Rebase should be used to keep commit history clean and avoid unnecessary merge commits in feature branches.


Git rebase
Reapply commits on top of another base commit
Integrate changes from one branch onto another
Linear commit history
Keep commit history clean
Avoid unnecessary merge commits

GIT Commands



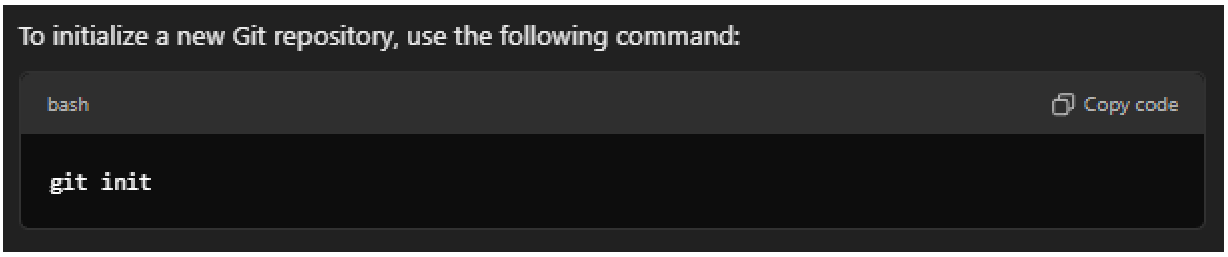
To set up your Git username and email, you can use the following commands:

1. **Set Global Username:**

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

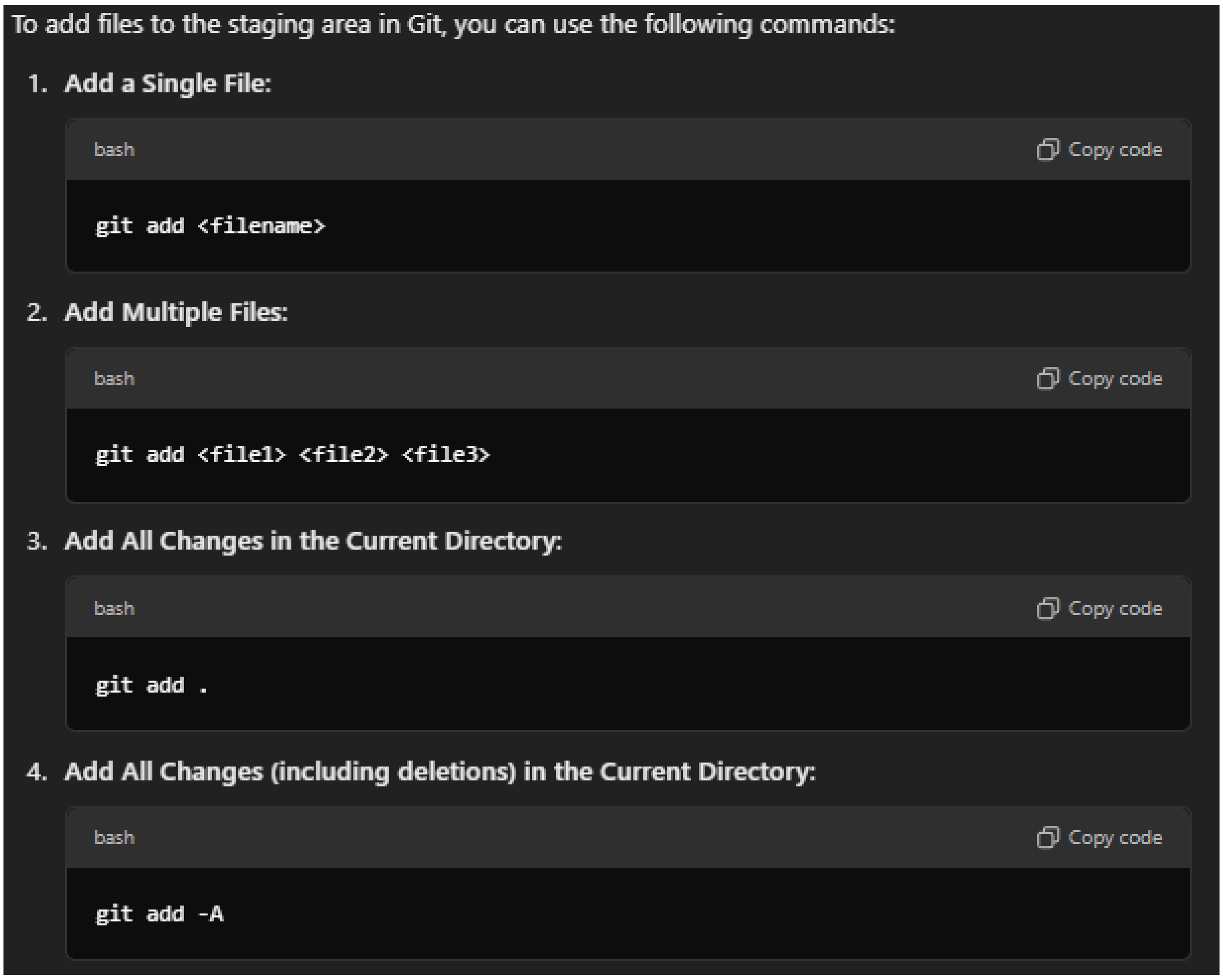
2. **Set Global Email:**

```bash
git config --global user.email "youremail@example.com"
```



To initialize a new Git repository, use the following command:

```bash
git init
```



To add files to the staging area in Git, you can use the following commands:

1. **Add a Single File:**

```bash
git add <filename>
```

2. **Add Multiple Files:**

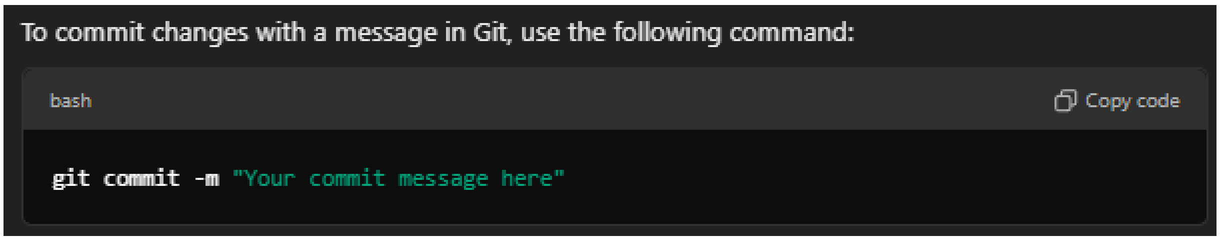
```bash
git add <file1> <file2> <file3>
```

3. **Add All Changes in the Current Directory:**

```bash
git add .
```

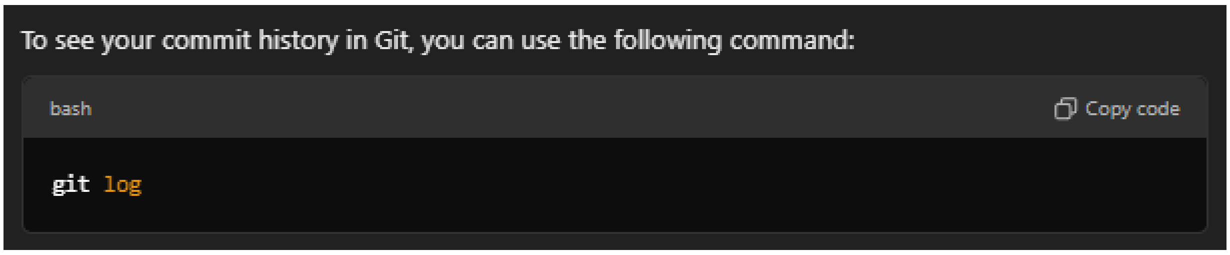
4. **Add All Changes (including deletions) in the Current Directory:**

```bash
git add -A
```



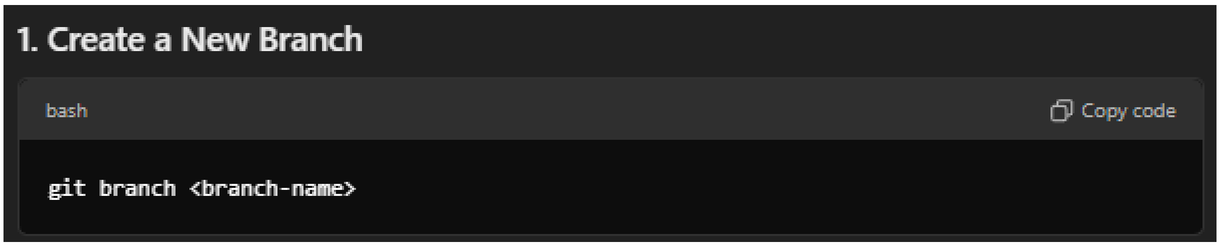
To commit changes with a message in Git, use the following command:

```bash
git commit -m "Your commit message here"
```



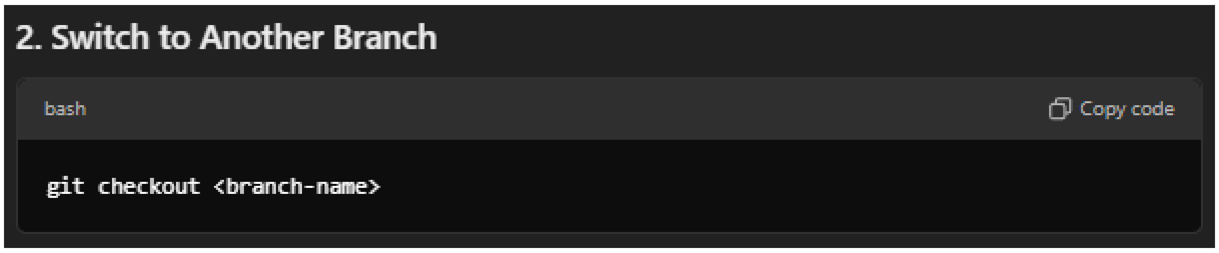
To see your commit history in Git, you can use the following command:

```bash
git log
```



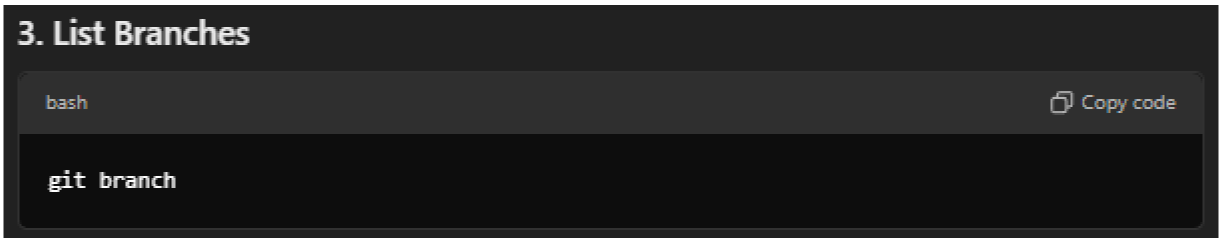
1. Create a New Branch

```bash
git branch <branch-name>
```



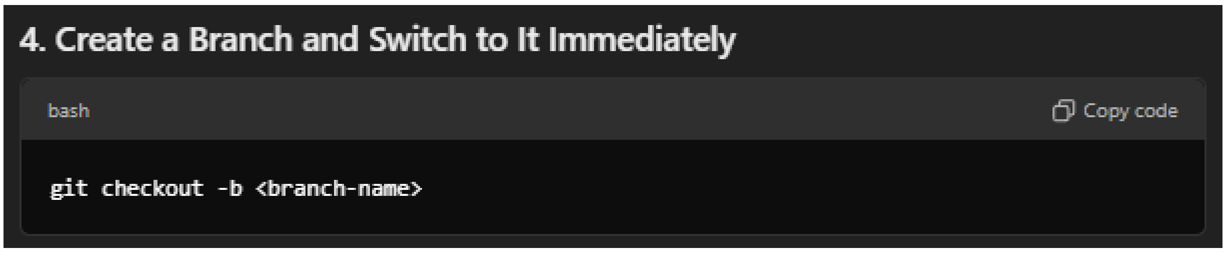
2. Switch to Another Branch

```bash
git checkout <branch-name>
```



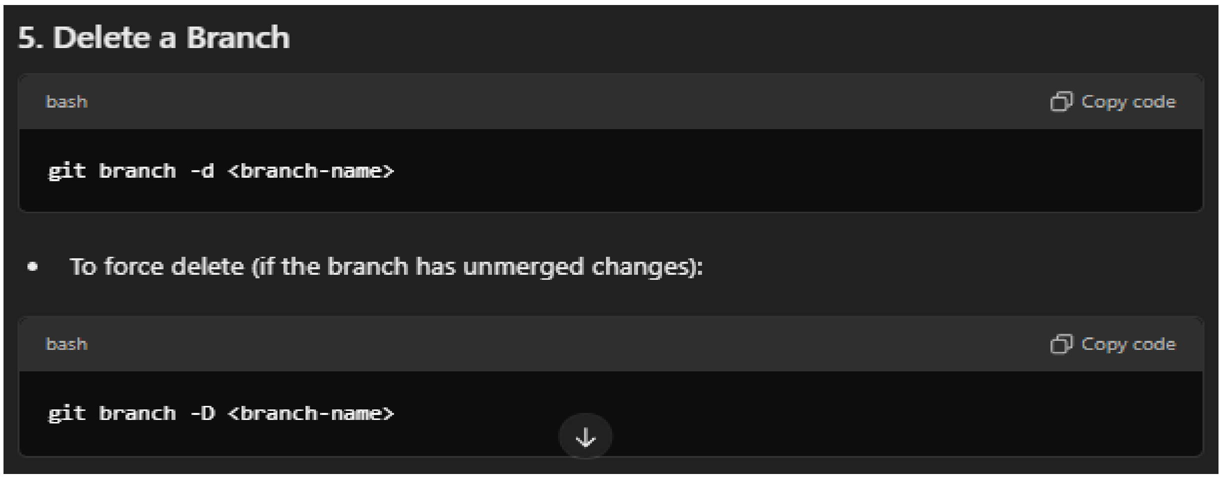
3. List Branches

```bash
git branch
```



4. Create a Branch and Switch to It Immediately

```bash
git checkout -b <branch-name>
```

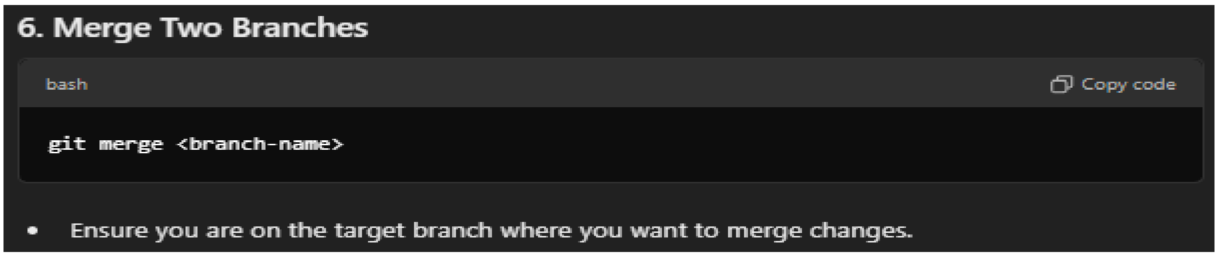


5. Delete a Branch

```bash
git branch -d <branch-name>
```

- To force delete (if the branch has unmerged changes):

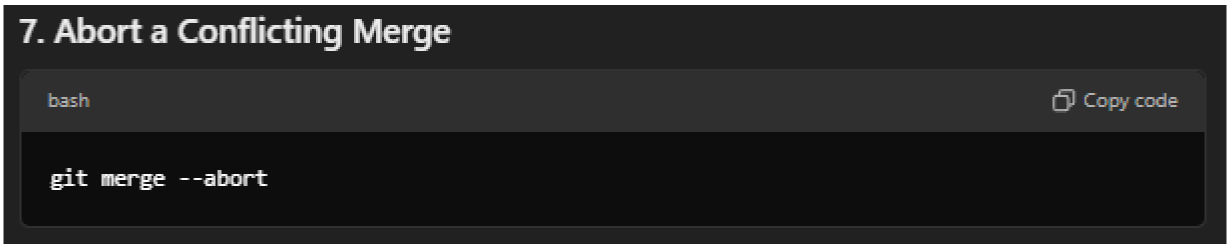
```bash
git branch -D <branch-name>
```



6. Merge Two Branches

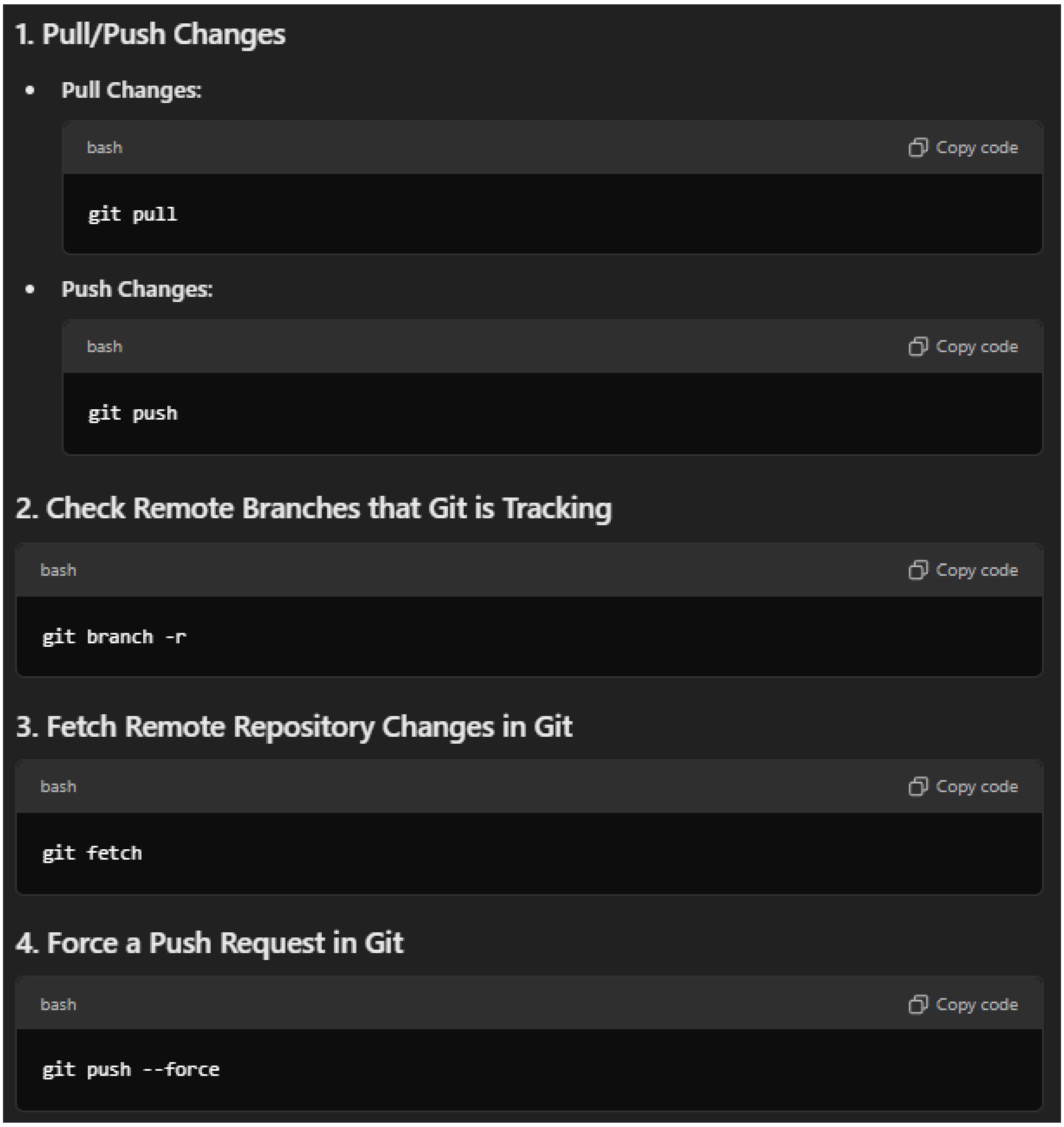
```bash
git merge <branch-name>
```

- Ensure you are on the target branch where you want to merge changes.



7. Abort a Conflicting Merge

```bash
git merge --abort
```



1. Pull/Push Changes

- **Pull Changes:**

  ```bash
  git pull
  ```

- **Push Changes:**

  ```bash
  git push
  ```

2. Check Remote Branches that Git is Tracking

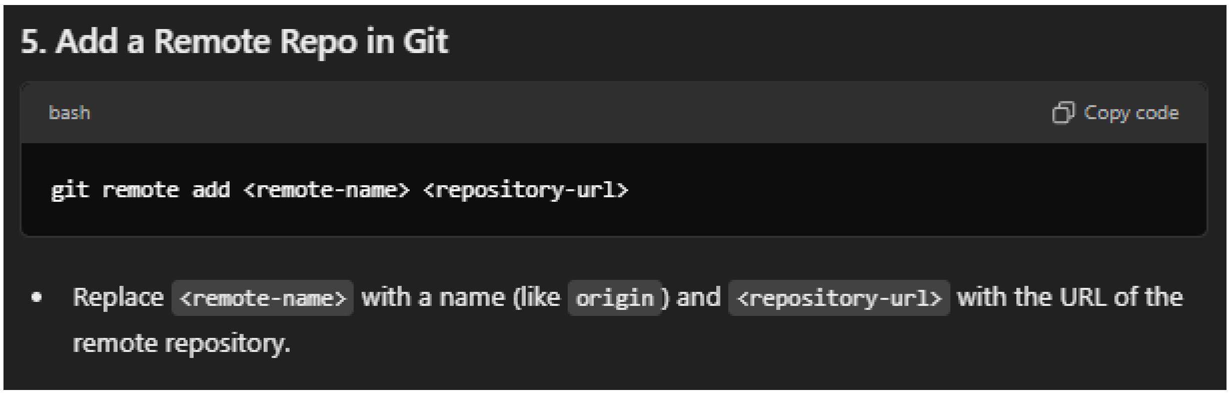
```bash
git branch -r
```

3. Fetch Remote Repository Changes in Git

```bash
git fetch
```

4. Force a Push Request in Git

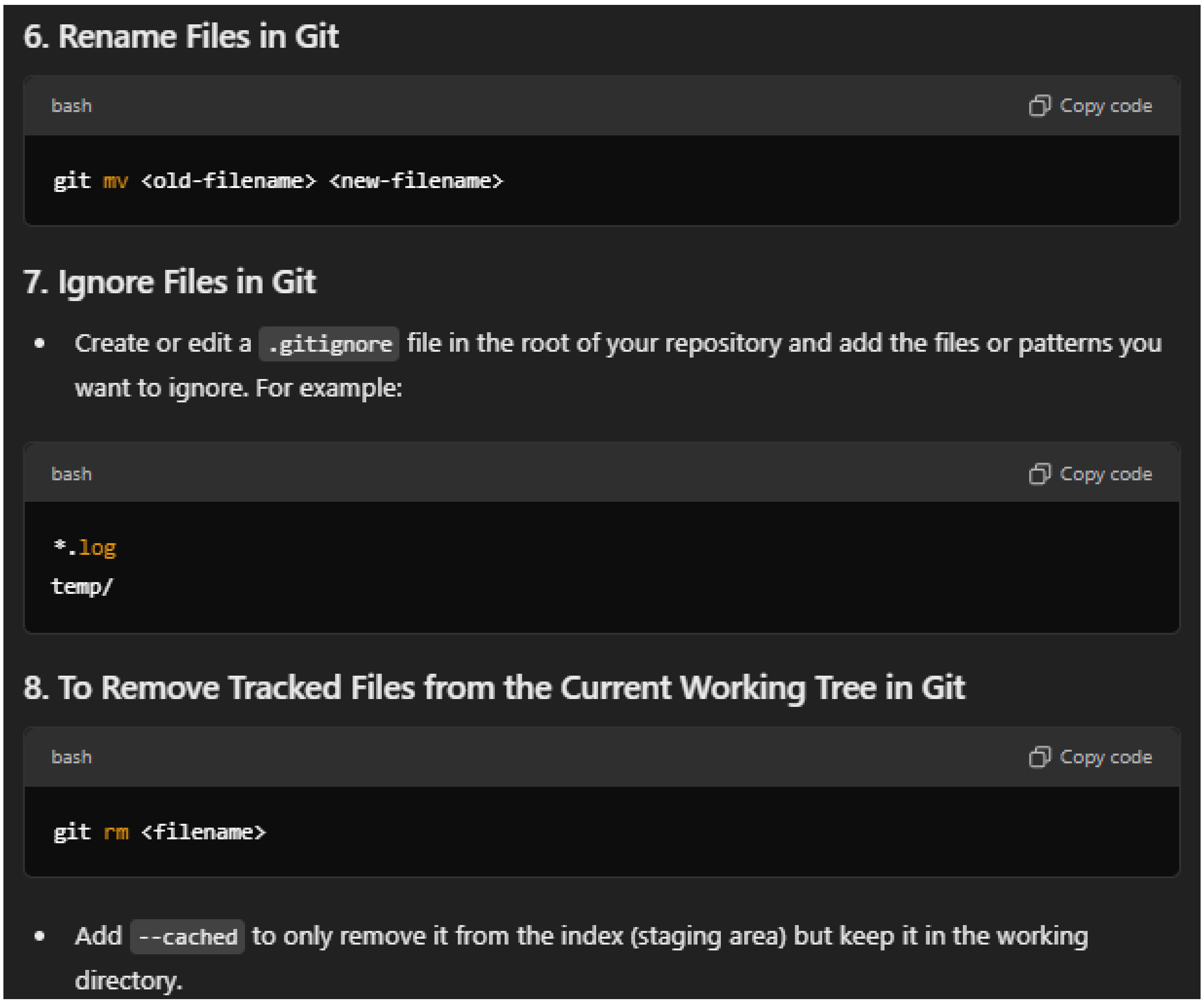
```bash
git push --force
```



5. Add a Remote Repo in Git

```bash
git remote add <remote-name> <repository-url>
```

- Replace `<remote-name>` with a name (like `origin`) and `<repository-url>` with the URL of the remote repository.



6. Rename Files in Git

```bash
git mv <old-filename> <new-filename>
```

7. Ignore Files in Git

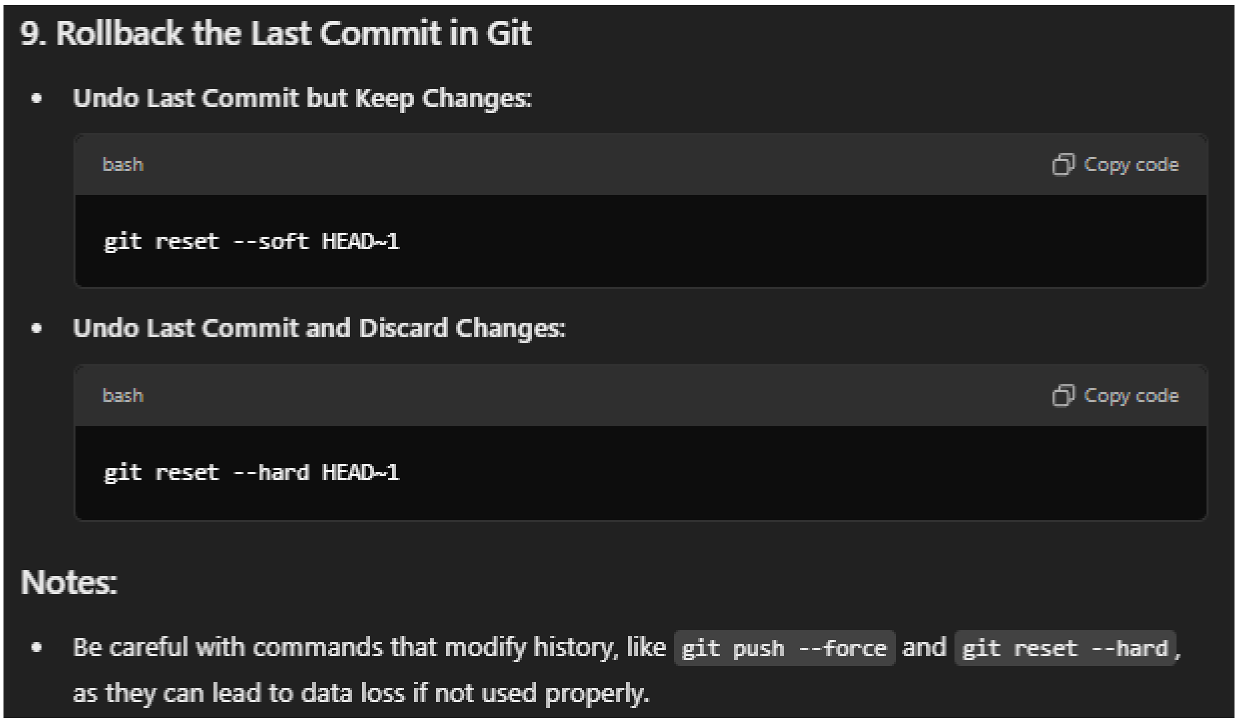
- Create or edit a `.gitignore` file in the root of your repository and add the files or patterns you want to ignore. For example:

```bash
*.log
temp/
```

8. To Remove Tracked Files from the Current Working Tree in Git

```bash
git rm <filename>
```

- Add `--cached` to only remove it from the index (staging area) but keep it in the working directory.



9. Rollback the Last Commit in Git

- **Undo Last Commit but Keep Changes:**

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

- **Undo Last Commit and Discard Changes:**

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

Notes:

- Be careful with commands that modify history, like `git push --force` and `git reset --hard`, as they can lead to data loss if not used properly.