Here are some essential Git commands and their use cases that every developer should know:

**1.**git init

* **Use Case**: Initializes a new Git repository in your project directory.
* **Example**:
* git init

**2.**git clone

* **Use Case**: Creates a copy of an existing repository from a remote server.
* **Example**:
* git clone https://github.com/user/repo.git

**3.**git status

* **Use Case**: Displays the state of the working directory and staging area, showing which changes have been staged, which haven’t, and which files aren’t being tracked by Git.
* **Example**:
* git status

**4.**git add

* **Use Case**: Adds changes in the working directory to the staging area.
* **Example**:
* git add filename
* git add .

**5.**git commit

* **Use Case**: Records changes to the repository with a descriptive message.
* **Example**:
* git commit -m "Commit message"

**6.**git log

* **Use Case**: Shows the commit history for the repository.
* **Example**:
* git log

**7.**git branch

* **Use Case**: Lists, creates, or deletes branches.
* **Example**:
* git branch
* git branch new-branch
* git branch -d old-branch

**8.**git checkout

* **Use Case**: Switches branches or restores working tree files.
* **Example**:
* git checkout branch-name
* git checkout -b new-branch

**9.**git merge

* **Use Case**: Merges changes from one branch into another.
* **Example**:
* git merge branch-name

**10.**git pull

* **Use Case**: Fetches and integrates changes from a remote repository into the current branch.
* **Example**:
* git pull origin main

**11.**git push

* **Use Case**: Uploads local repository content to a remote repository.
* **Example**:
* git push origin main

**12.**git stash

* **Use Case**: Temporarily saves changes that are not ready to be committed.
* **Example**:
* git stash
* git stash pop

**13.**git remote

* **Use Case**: Manages set of tracked repositories.
* **Example**:
* git remote add origin https://github.com/user/repo.git
* git remote -v

**14.**git fetch

* **Use Case**: Downloads objects and refs from another repository.
* **Example**:
* git fetch origin

**15.**git rebase

* **Use Case**: Reapplies commits on top of another base tip.
* **Example**:
* git rebase branch-name

**Steps to Resolve Merge Conflicts**

1. **Identify the Conflict**
   * When you try to merge branches and a conflict occurs, Git will notify you. Use git status to see which files have conflicts.
2. git status
3. **Open the Conflicted Files**
   * Open the files with conflicts in your favorite text editor. Conflicted sections will be marked with conflict markers:
4. <<<<<<< HEAD
5. // Your changes
6. =======
7. // Changes from the other branch
8. >>>>>>> branch-name
9. **Resolve the Conflicts**
   * Decide how to resolve the conflicts. You can choose to keep your changes, the other branch’s changes, or a combination of both. Edit the file to remove the conflict markers and make the necessary changes.
10. **Mark the Conflict as Resolved**
    * After resolving the conflicts, mark the file as resolved using git add.
11. git add filename
12. **Commit the Changes**
    * Commit the resolved changes to complete the merge.
13. git commit -m "Resolved merge conflict in filename"
14. **Continue the Merge**
    * If you were in the middle of a merge, you can continue the merge process.
15. git merge --continue

**Using Git Mergetool**

You can also use a merge tool to help resolve conflicts. Git supports several merge tools like vimdiff, meld, kdiff3, etc.

1. **Configure the Merge Tool**
   * Set up your preferred merge tool.
2. git config --global merge.tool meld
3. **Run the Merge Tool**
   * Use the merge tool to resolve conflicts interactively.
4. git mergetool

**Example of Resolving a Conflict**

Suppose you have a conflict in a file named example.txt. The conflict markers might look like this:

<<<<<<< HEAD

Your changes here

=======

Changes from the other branch

>>>>>>> branch-name

You can resolve the conflict by editing the file to remove the markers and combine the changes as needed:

Combined changes from both branches

Then, mark the file as resolved and commit the changes:

git add example.txt

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

To merge all changes from one branch into another as a single commit, you can use the --squash option with the git merge command. This approach combines all the changes from the feature branch into a single commit on the target branch. Here’s how you can do it:

**Using**git merge --squash

1. **Checkout the target branch**:
2. git checkout main
3. **Merge the feature branch with the**--squash**option**:
4. git merge --squash feature-branch
5. **Commit the changes**:
6. git commit -m "Merged feature-branch as a single commit"