**6.1 Understanding Common Issues in Git**

**1. Merge Conflicts** Occurs when Git cannot automatically resolve differences between two branches.Common causes: **🡪**Multiple developers modify the same line of a file. **🡪**Differences in file structure between branches. **2. Detached HEAD** Happens when you checkout a specific commit, not a branch.Changes made in this state are not associated with a branch unless explicitly saved. **3. File Deletions** Unintentionally deleting files or commits that need recovery. **4. Out-of-Sync Branch** When your branch falls behind the remote branch, causing conflicts during pull or merge.

**6.2 Tools for Debugging in Git  
6.2.1 git log** Explore commit history to debug when a specific change was introduced. **Commands**:

1. View the history:🡪 git log
2. See concise logs:🡪 git log --oneline
3. Search for specific commits:🡪 git log --grep="fix bug"

**6.2.2 git blame**  
 Shows who made changes to specific lines in a file.  
**Command**:  
 git blame <file>  
**Use Case**:  
 Identify the commit and author responsible for a particular line of code.

**6.2.3 git bisect**  
 A binary search tool to find the commit that introduced a bug.  
**Steps**:

1. Start bisect:🡪 git bisect start
2. Mark the current commit as bad:🡪 git bisect bad
3. Mark a known good commit:🡪 git bisect good <commit-hash>
4. Git checks commits between the good and bad states. Test each commit until the buggy one is identified.
5. Reset after finding the issue:🡪 git bisect reset

**6.2.4 git reflog**  
 Tracks changes made to HEAD, even if commits are deleted or unreachable.  
**Command**:

git reflog  
**Use Case**:  
 Recover lost commits by identifying their hashes.

**6.2.5 git diff**  
 Compare changes between commits, branches, or working states.  
**Commands**:  
 Compare working directory and staging area:🡪 git diff  
Compare two commits:

git diff <commit1> <commit2>

1. Compare current branch with another branch:🡪 git diff <branch1> <branch2>

**6.2.6 git show** Displays details of a specific commit, including changes. **Command**:git show <commit-hash> **6.2.7 git fsck** Verifies the integrity of the repository and identifies broken references. **Command**:git fsck

**6.3 Resolving Issues in Git**

**6.3.1 Resolving Merge Conflicts**

1. Identify conflicting files:🡪 git status
2. Open the files and resolve conflicts manually:

<<<<<<< HEAD

Code from your branch

=======

Code from other branch

>>>>>>> other-branch

1. Add resolved files:  
    git add <file>
2. Complete the merge:  
    git commit

**6.3.2 Recovering Deleted Files**

1. Recover from the last commit:  
    git checkout HEAD -- <file>
2. Recover a file from a specific commit:  
    git checkout <commit-hash> -- <file>

**6.3.3 Undoing Commits**

1. **Undo the last commit but keep changes:**  
    git reset --soft HEAD~1
2. **Undo the last commit and discard changes:**  
    git reset --hard HEAD~1

**6.3.4 Fixing Detached HEAD**

1. Create a new branch from the detached HEAD:  
    git checkout -b <new-branch-name>

**6.3.5 Handling Out-of-Sync Branches**

1. Rebase your branch with the latest changes:  
    git pull --rebase origin main
2. Resolve conflicts, if any, and complete the rebase:  
    git rebase –continue

**6.4 Preventive Best Practices**

1. **Commit Often**:
   * Commit small, logical changes to minimize conflicts.
2. **Pull Regularly**:
   * Keep your branch in sync with the remote:  
      git pull origin main
3. **Use Meaningful Commit Messages**:
   * Descriptive commit messages help in debugging.
4. **Avoid Force Push**:
   * Unless necessary, avoid rewriting shared history.
5. **Use Branching Strategies**:
   * Isolate features and bug fixes in separate branches.
6. **Verify Changes Before Committing**:
   * Use git diff to check changes.