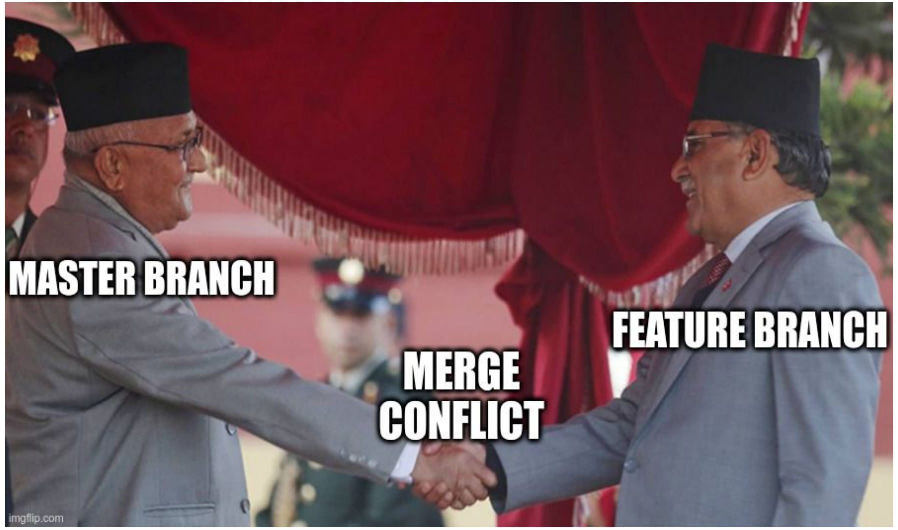
MERGE CONFLICTS AND REVERTING CHANGES

MERGE CONFLICTS

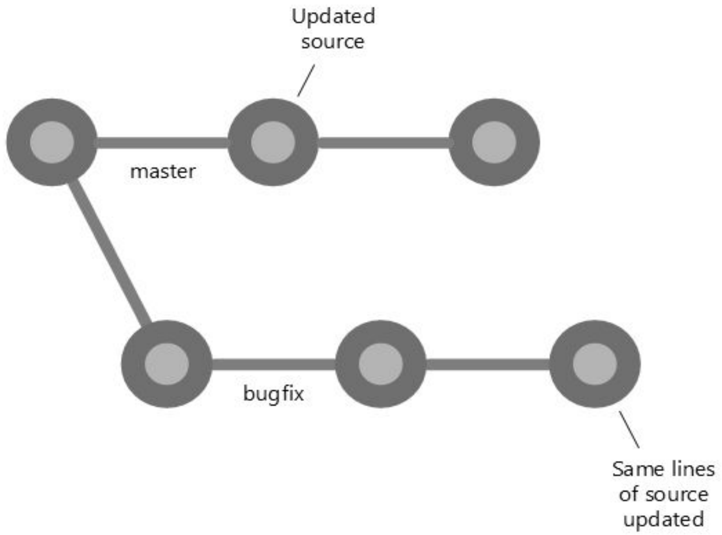
What is Merge Conflict ?

A merge conflict is an event that takes place when Git is unable to automatically resolve differences in code between two branches.



How does it occur ?

It occurs when two people on different branches have modified the same lines of code.



What does it affect ?

It affects the branch that is trying to merge the other branches. The other branches are as it is.

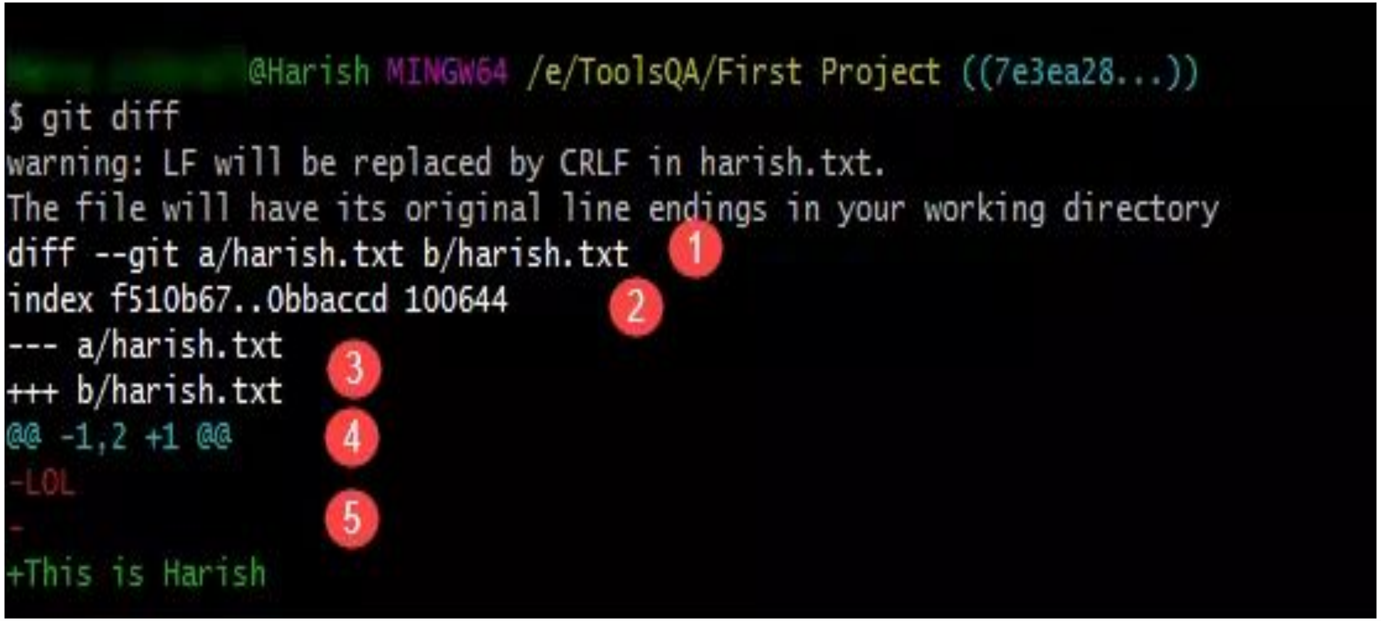
Identifying and Resolving Conflicts?

Git issues merge failed error when such branches are tried to merge. To resolve the merge conflicts, we should manually handle the commits.

COMPARING BRANCHES

Using Diff Command

Review: used to compare contents of different commits and branches as well.



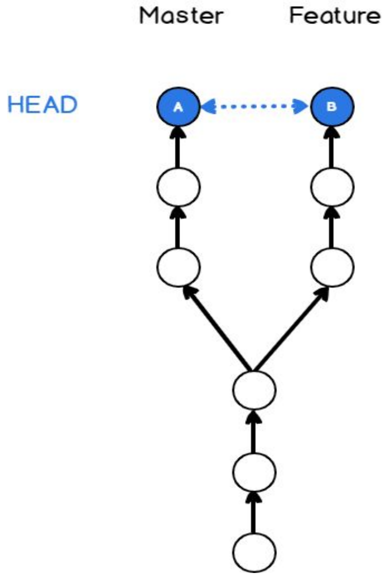
```
[redacted]@Harish MINGW64 /e/ToolsQA/First Project ((7e3ea28...))
$ git diff
warning: LF will be replaced by CRLF in harish.txt.
The file will have its original line endings in your working directory
diff --git a/harish.txt b/harish.txt
index f510b67..0bbaccd 100644
--- a/harish.txt
+++ b/harish.txt
@@ -1,2 +1 @@
-LOL
-
+This is Harish
```

Some details provided by the diff command

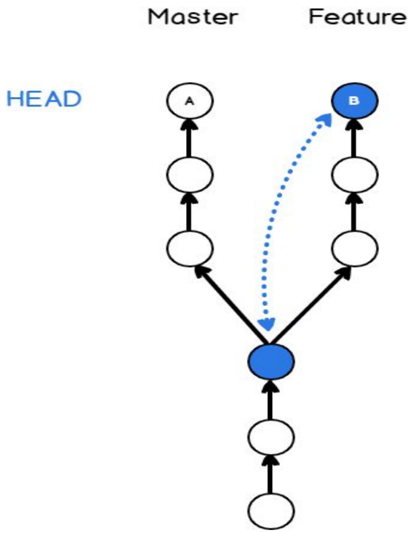
1. Compared files a/b
2. File Metadata
3. Markers. --- for indicating lines removed. +++ for lines added.
4. Chunk header.
5. Changes.

How to compare between branches?

git diff master..feature



git diff master...feature
triple dot



ANY QUESTIONS ??

MOVING AROUND COMMITS

Some Basic Terms

REFS
Is an indirect way of referring to a commit. User friendly alias for commit hash.

HEAD
It points to the most recent commit reflected in the working tree.

RELATIVE REFS
We can use relative position between branch pointer or HEAD to move around different commits with the help of what is called as relative references.

Commands and Operators used:

- **Checkout command** is used to switch between branches as well as between commits
- ^ operator specifies the parent of the current ref and ^^ specifies the grandparent of the current ref
- ~<num> specifies the ref ‘num’ number of nodes above the tree.

Note : After checking out to some commit, HEAD changes to some commit instead of main(master), this process is called detaching HEAD.

What can we do with this ?

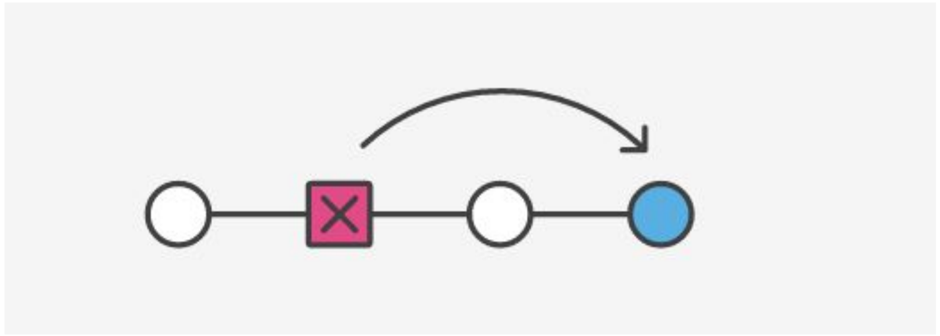
- We can move to a different commit and then check the contents of file and activities of that commit.

- We can restore some file from the last commit.
  git checkout HEAD~2 index.html
  Here, index.html file is retrieved from grandparent of HEAD. (be careful)

REVERTING CHANGES

Revert command

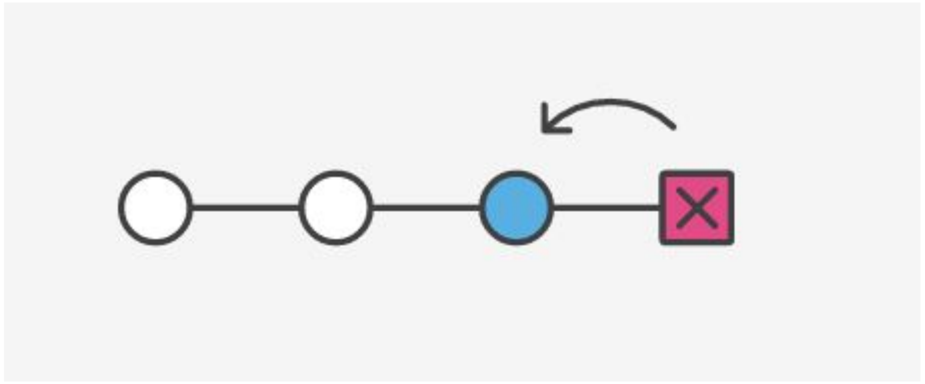
Condition : Suppose you are working in a branch, and suddenly remember that you need the contents from previous commit into this commit without changing the history of the commits. How to do this?



- You are in luck, **git revert** is exactly used for this. Let’s see this in our example.

Reset command

Revert command doesn’t "revert" back to the previous state of a project by removing all subsequent commits. For this, reset command is to be used.



Types of resets

1. Soft Reset

   Removes the commit. But the files are at the staging areas.

   $ git reset --soft <commit_hash>

2. Mixed Reset

   This is the default reset mode. Removes the staged files from staging to working area.

   $ git reset <commit_hash>

3. Hard Reset

   Completely resets to the previous commit.

   $ git reset --hard <commit_hash>

Points to Remember:

- Hard reset is not reversible so use it only when needed.
- Untracked files are not affected by revert and reset. To remove untracked files **git clean** command is used.

Amending changes

Condition

If you ever are in a situation where you committed a change, but the change that you committed was not complete, i.e you wanted something to add to that same commit, then amend flag comes to the rescue.

```
$ git commit --amend
```

ANY QUESTIONS ??