**21)Detached Head in GIT**

When we are working, we only checkout one branch at a time. This is also called the Head branch.

Git makes note of this branch and stores it in .git/Head as the reference for the path of the branch

Head is nothing but the reference to the branch.

Head not only references a branch, it also references the commit sha1.

If the head points to a specific commit then it is called as detached head.

A picture containing diagram

Description automatically generated

We start by git checkout e137e9b not git checkout master

In the above Head is pointing to a particular commit. Master is also pointing towards a particular commit.

Now when we do git commit

Diagram

Description automatically generated

The head moves to the next commit but not the master, the commit does not belong to any branch.

Diagram

Description automatically generated with medium confidence

The same happens if we do another commit.

Now if we do git checkout master the head moves to master.

Diagram

Description automatically generated

The two commits will be garbage collected until we know the commits hashcodes.

Now if we do git commit the new commit will be added to master branch

A picture containing text, electronics

Description automatically generated

We can add the two left-over commits by using the hash code

git checkout 79ac38e now the last commit becomes head

Diagram

Description automatically generated

then we can do git branch feature, now the two commits will be added to a new branch named feature.

A picture containing text, electronics

Description automatically generated

**22)Git Reset**

The term reset itself stands for undoing changes. Reset is often referred to as confusing command. Reset does different things in different contexts. So that's why it is considered a complex command.

We know that if you want to move the branch, we use following commands

1. Commit

2. Merge

3.Rebase

If you observe, these commands are not explicitly used for moving the branch. The branches are moved as a side effect of creating new commit.

Do we have any command that is specially used for moving a branch?

Reset does this. It is used to move the branch.

**Reset has options**

--hard -> moves the files both to working area and staging area

--mixed - moves the files only to stage area. (Default option)

--soft- does not move the files

We can say that Reset moves the current branch and optionally copies the data from the repositories to the working or staging areas.

A picture containing diagram

Description automatically generated

After we do two commits

A picture containing text

Description automatically generated

Diagram

Description automatically generated

To move head

Text

Description automatically generated

Diagram

Description automatically generated

To again move head

Text

Description automatically generated

Diagram

Description automatically generated

Now if we do git reset e159164 the master and head will move to previous commit (Assuming hard)

Diagram

Description automatically generated with medium confidence

Practical use is to undo a feature we can use git reset.

**23)Git Reset Practical Usage**

Let’s create a project and make a file index.html(committed) in master branch.

Let’s create a new feature and do 3 commits in the master branch. Now the head in master branch moves three steps. We added feature.txt file (committed).

But for some reason we no longer need this feature then we can move back to 1st stage where there is only index.txt.

Text

Description automatically generated

Move master from b22767d to 304192b.

git reset 304192b

The two commits b22767d and ad02a23 are removed but feature file still exists.

But if we do git reset 304192b -- hard the feature file will also be removed

**Another usage**

Let’s say a file index.txt is added to staging area

git add. Adds all files to staging area.

Now we can revert the file to working area by

git restore --staged

or

git reset head –mixed

But

git reset head --hard removes files from working area and staging area also.

But

git reset head –soft wont remove any files from anywhere it changes moves the commit

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

**24)Git Stashing**

Sometimes you want to switch branches, but you are working on an incomplete part of your current project.

You don't want to make a commit of half-done work. Git stashing allows you to do so.

Normally when you switch branch you will commit the code and switch to the new branch.

If you switch branches without committing.

**Two things will happen.**

1. Switches to the branch carrying the changes. (We will commit in that other branch)

2. Git will not allow to switch the branch and asks to commit or stash the changes

The git stash command enables you to switch branches without committing the current branch.

The stash's meaning is "store something safely in a hidden place." The sense in Git is also the same for stash; Git temporarily saves your data safely without committing.

Let’s say in a project we have master branch in which we committed an index file.

In another branch named feature we are working on file named featurefile and we need to shift to master branch, and we are not done working with the featurefile.

This is where stashing comes into play.

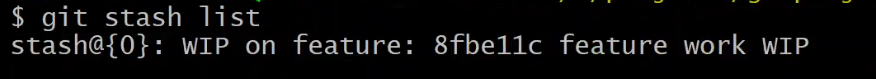
**25)Git Stashing, Save and Retrieve Stash Data**

So, we stash the featurefile. We do that by

git stash

To make sure whether the file went to stash area or not we can check by,

git stash list This gives list of all stash files



Now we can checkout to other branches.

After some time, we can return to feature branch and we can get back the stashed file by,

git stash pop This gives us back the file.

Now if we check the stash list it’ll be empty.

We can save the file in stash with a custom name by,

git stash save "modified index"

Text

Description automatically generated

Popping a file from stash list will automatically delete it from stash list.

If we want to retrieve from stash but not want to delete it from stash we have to,

git stash apply stash@{0} (stash@{0} id of stash)

If we do git stash apply (The most recent stash will be saved)

Text

Description automatically generated

We can stash a file in one branch and apply it in another branch.

**26)Handle Multiple Stashes, Delete and clear Stash data**

git stash show stash@{1} That file will be used

git stash show Most recent will be used

Text

Description automatically generated

git stash show is used to see the changes in between the stashed files.

Above we can see that 1 line has been added.

git stash show -p We can see the changes line by line

Text

Description automatically generated

git stash drop stash@{1} That stash will be deleted from stash list

git stash drop Most recent stash will be deleted from stash list

git stash clear All files in stash list are deleted

git stash branch new\_feature stash@{0}

A new branch named new\_feature is created with that stash file

**27)Git Checkout, Different Ways of Using Checkout**

Git checkout does lot of things

**Moves from one branch to other**

git checkout feature It moves to that branch

**Creates new branch if not existed and moves head to that branch**

git checkout -b new\_feature creates a branch named new\_feature.

**Also not only branch it also shifts to a particular commit hash if you observe**

git checkout d376341head moves to that commit

**To go to previous state**

Now the head is in **d376341**

git checkout – makes the head go to previous state

the head came to **d376341** from master branch (in this case) so

git checkout – makes head move to master branch

**To create a new branch from a commit**

git checkout d376341

git checkout -b new\_branch

This makes a new branch named new\_branch from the commit d376341

We can return to master by git checkout master

**Move Back Head**

git checkout Head~n

Text

Description automatically generated

Head is at da84a57

git checkout~2 now head goes two commits behind to

c495581

Means without knowing the hash we can move the commit n-steps back

**Previous Position**

If you want to go back where you are previous means git checkout –

Previous position of head

**Git checkout also used to discard changes in the file**

You can revert the changes of the particular file using

git checkout Head <filename>

or

git checkout -- <filename>

Let’s say all files are committed now let’s edit two files index.txt and master.txt.

git reset --hard (All changes are reverted) even from working directory

But if we want to revert change in one file only

git checkout head index.txt (index.txt file changes are reverted, other file name not reverted)

git checkout -- index.txt (index.txt file changes are reverted, other file name not reverted)

**28)Restore and Switch Command**

**Switch Command**

**To move head from one branch to another**

Graphical user interface, text, application, chat or text message

Description automatically generated

git branch we get all branches

git switch feature head moves from master to feature

**To create new branch**

git switch -c new\_branch2 creates a new branch named new\_branch2

**Switch can only take branches not commits**

checkout can go to a particular commit, but switch cannot, it can only go to a branch

git checkout~2 works

git checkout~2 does not work

**Cannot revert changes like checkout**

Suppose there is a file we made changes and added to staging area

To same file we did another change this is in working directory

git checkout head index.txt all changes reverted (both working and staging)

**Restore**

**To move file from staging to working area**

git restore -- staged index.txt

To remove changes in working area only we can use restore

git restore index.txt

**To make current file state previous state’s**

Text

Description automatically generated

If we want index.txt file to look like how it looks at c495581at current position

git restore --source Head~2 index,txt (2 because da84a57 2 steps ahead)

**29)Revert, Revert vs Reset**

Revert is used to revert a commit i.e., if we mistakenly committed some files, we could use revert to revert the changes.

In master branch two files index and feature files (Committed)

Now we make some changes to index file and commit again.

Now we make some changes to feature file and commit again.

Now we make some changes to index file and commit again.

Now we make some changes to feature file and commit again.

Text

Description automatically generated

If we want to remove last commit

git reset – hard Head~1 Head moved one step back

Text

Description automatically generated with medium confidence

Reset has no history that there was one more commit.

**Revert**

In master branch two files index and feature files (Committed)

Now we make some changes to index file and commit again.

Now we make some changes to feature file and commit again.

Now we make some changes to index file and commit again.

Now we make some changes to feature file and commit again.

Text

Description automatically generated

git revert f65bc14 this reverts last commit(It asks for a message)

Text

Description automatically generated

Revert command keeps history that we reverted one commit reset doesn’t do it.

Reset is used for local

Revert is used when working with others.

Text

Description automatically generated

In the above we undo the revert command.

The main use of revert is we can revert changes of master file changes 1 commit without deleting the above 2 commits (reset cant do it)(f65bc14,0f7cca5)

We can do

git revert 7d962f1

Text

Description automatically generated