**DEVOPS**

**GIT REPOSITORY STEPS**

**Git Tutorial – Operations & Commands**

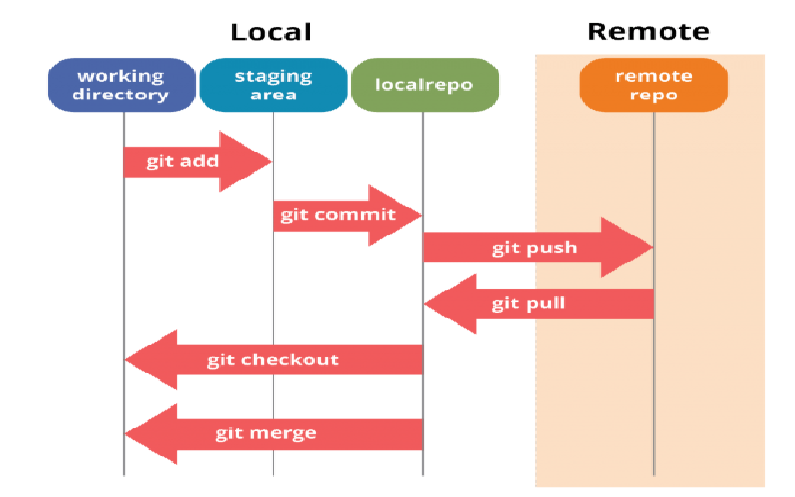
**Some of the basic operations in Git are:**

1. Initialize
2. Add
3. Commit
4. Pull
5. Push

**Some advanced Git operations are:**

1. Branching
2. Merging
3. Rebasing

**First let me give you brief idea about how these operation works with Git repositories.**

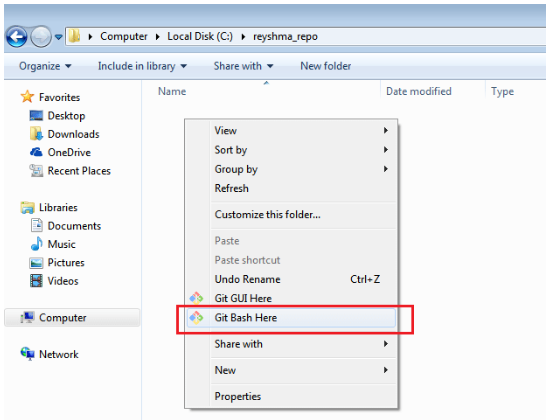
****

If you understand the above diagram well and good, but if you don’t, you need not worry, I will be explaining these operations in this Git Tutorial one by one. Let us begin with the basic operations.

You need to install Git on your system first. If you need help with the installation, <https://www.edureka.co/blog/install-git/>

In this Git Tutorial, I will show you the commands and the operations using Git Bash. Git Bash is a text-only command line interface for using Git on Windows which provides features to run automated scripts.

After installing Git in your Windows system, just open your folder/directory where you want to store all your project files; right click and select ‘***Git Bash here***’.

****

This will open up Git Bash terminal where you can enter commands to perform various Git operations.

Now, the next task is to initialize an empty repository.

## ****Initialize****

## **In order to do that we have to use the command **git init** , please refer the below screen shot.**

## The git init command creates the new empty repository or re-initializes the old one, basically it creates .git directory with subdirectory and templates. Running git init in an existing git repository it will not over write changes are already there, it rather just pick up the newly added templates.

Now that my repository is initialized, let me create some files in the directory/repository. For e.g. I have created two text files namely rami1.txt, ram2i.txt using **touch** command.



Let’s see if these files are in my index or not using the command **git status**. The index holds a snapshot of the content of the working tree/directory, and this snapshot is taken as the contents for the next change to be made in the local repository.

**Git status**

The **git status**command lists all the modified files which are ready to be added to the local repository.

Let us type in the command to see what happens:

## 

## This shows that I have two files which are not added to the index yet. This means I cannot commit changes with these files unless I have added them explicitly in the index.

**Add**

This command updates the index using the current content found in the working tree and then prepares the content in the staging area for the next commit.

Thus, after making changes to the working tree, and before running the **commit** command, you must use the **add** command to add any new or modified files to the index. For that, use the commands below:

**git add <directory>**

or

**git add <file>**

Let me demonstrate the **git add** for you so that you can understand it better.

I have created two more files edureka3.txt and edureka4.txt. Let us add the files using the command **git add -A**. This command will add all the files to the index which are in the directory but not updated in the index yet.

## 

## Now that new files added to the index, your ready to commit them using commit command.

**Commit**

It refers to recording snapshots of the repository at a given time. Committed snapshots will never change unless done explicitly. Let me explain how commit works with the diagram below:

## 

Here, C1 is the initial commit, i.e. the snapshot of the first change from which another snapshot is created with changes named C2. Note that the master points to the latest commit.

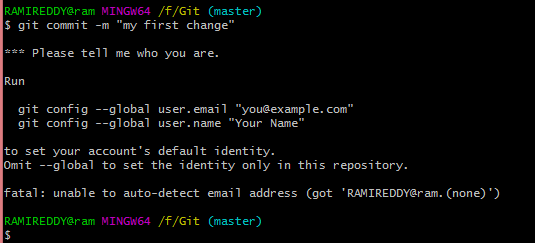
Now, when I commit again, another snapshot C3 is created and now the master points to C3 instead of C2.

Git aims to keep commits as lightweight as possible. So, it doesn’t blindly copy the entire directory every time you commit; it includes commit as a set of changes, or “delta” from one version of the repository to the other. In easy words, it only copies the changes made in the repository.

You can commit by using the command below:

**git commit**

**This command will commit your changes to local repository from staging snapshot and launch the text editor prompting you for a commit message.**

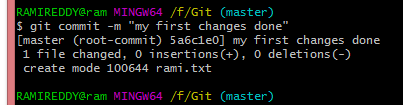


**While committing if you are facing the above error then run the following command to sort out from the above error.**

**git config –global user.name “rami reddy”**

**git config –global user.email “**[ramireddyvakamalla@gmail.com](mailto:ramireddyvakamalla@gmail.com)**”**

**After you issued above two commands then git allows you to commit changes staging to local repository.**



As you can see above, the **git commit** command has committed the changes in the four files in the local repository.

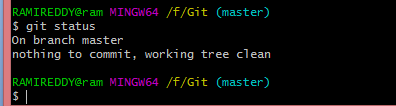
Now, if you want to commit a snapshot of all the changes in the working directory at once, you can use the command below:

**git commit –a “message”**

I have created two more text files in my working directory viz. edureka5.txt and edureka6.txt but they are not added to the index yet.

I am adding edureka5.txt using the command:

**Note: After did commit you can check using command git status whether changes to be made in local repository or not**



**NOTE:**  make sure that before you affect all changes to central repository, you should always pull changes from central repository to local repository to get updated with the work of all the collaborators that have been contributing the central repository. For that we will use the **PULL**  command.

## ****Pull****

## **The **git pull** command fetches all changes from central repo to local repository and it merges up-stream changes in our local repository. This is the common task in Git Collaboration.**

## ****But,** First you need to set central repository as **origin** using command below.**

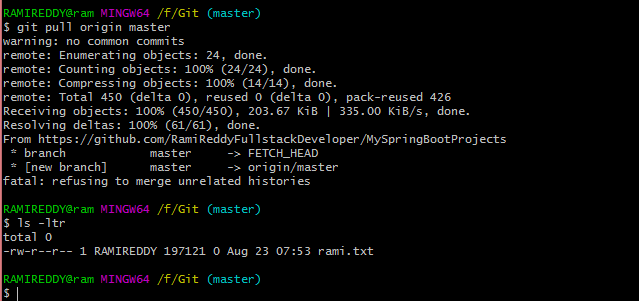
## git remote add origin <link of your central repository>

## 

Now that my origin is set, let us extract files from the origin using pull. For that use the command:

**git pull origin master**

This command will copy all the files from the master branch of remote repository to your local repository.



**Note:** This note just for my understanding purpose only. If you see above fatal error, that came due to I’m used un-related repository URL. If you use correct URL means working repository url then you will not see the fatal error.



**Note:** If use correct url you definitely see the above info.

Since my local repository was already updated with files from master branch, hence the message is Already up-to-date. Refer to the screen shot above.

Your local Git repository is now updated with all the recent changes from central repo. It is time you make changes in the central repository by using the**push** command.

**Push**

This command transfers commits from your local repository to your remote repository. It is the opposite of pull operation.

Pulling imports commits to local repositories whereas pushing exports commits to the remote repositories .

The use of **git push** is to publish your local changes to a central repository. After you’ve accumulated several local commits and are ready to share them with the rest of the team, you can then push them to the central repository by using the following command:

**git push origin <branch-name>**

**Note**: This remote refers to the remote repository which had been set before using the pull command.

This pushes the changes from the local repository to the remote repository along with all the necessary commits and internal objects. This creates a local branch in the destination repository.

To prevent overwriting, Git does not allow push when it results in a non-fast forward merge in the destination repository.

**Note**: A non-fast forward merge means an upstream merge i.e. merging with ancestor or parent branches from a child branch.

To enable such merge, use the command below:

**git push <remote> –force**

The above command forces the push operation even if it results in a non-fast forward merge.

At this point of this Git Tutorial, I hope you have understood the basic commands of Git. Now, let’s take a step further to learn **branching** and **merging** in Git.

**Branching**

Branches in Git are nothing but pointers to a specific commit. Git generally prefers to keep its branches as lightweight as possible.

There are basically two types of branches viz. ***local branches*** and ***remote tracking branches***.

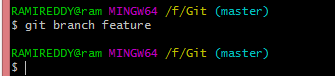
A local branch is just another path of your working tree. On the other hand, remote tracking branches have special purposes. Some of them are:

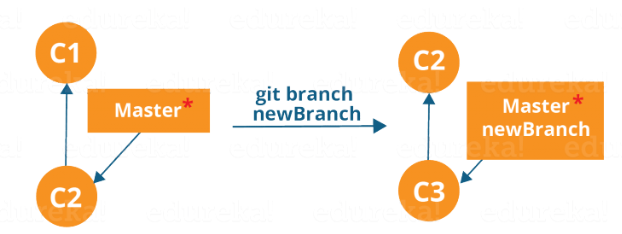
You can check what your current branch is by using the below command:

**git branch**

To create a new branch we use the following command:

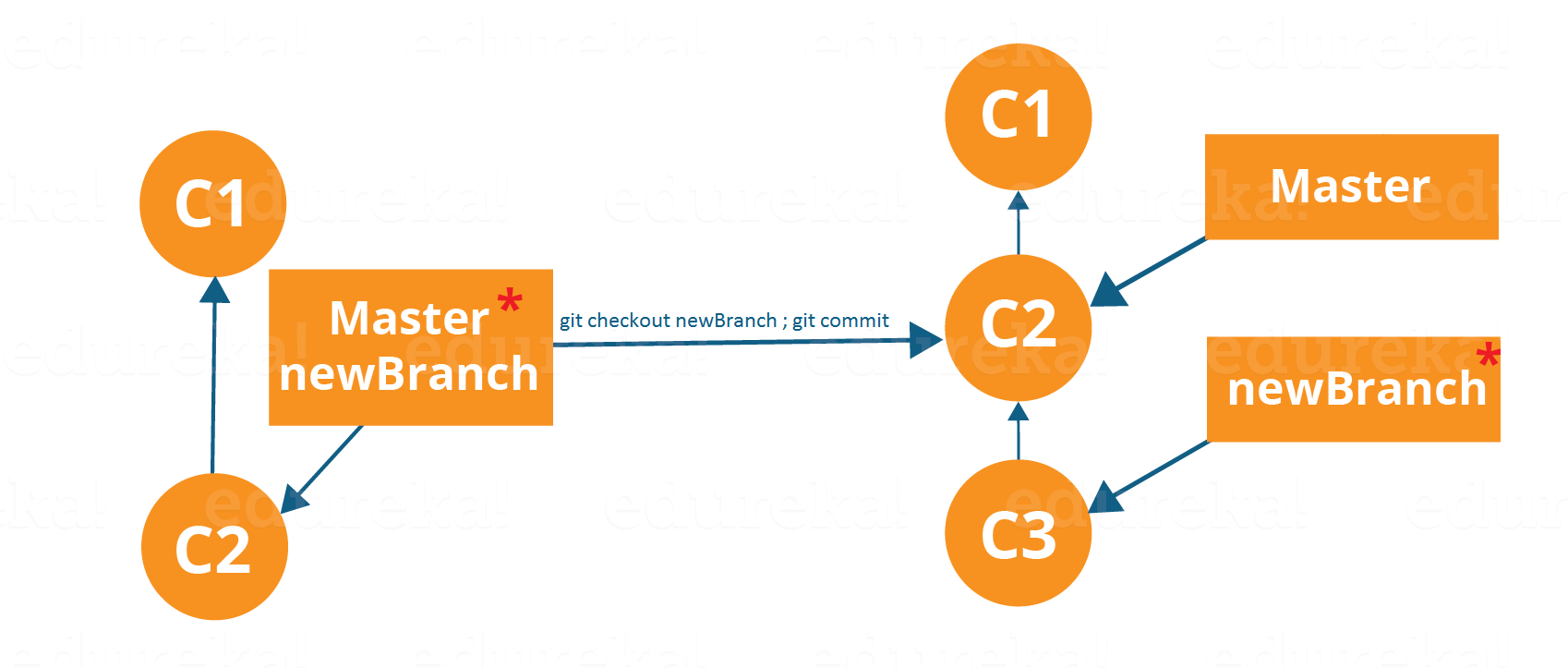
**git branch <branch-name>**





The diagram above shows the workflow when a new branch is created.  When we create a new branch it originates from the master branch itself.

Since there is no storage/memory overhead with making many branches, it is easier to logically divide up your work rather than have big chunky branches.

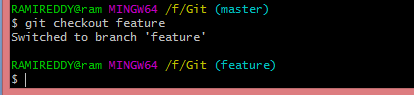
Now, let us see how commit using branches.

Branching includes the work of a particular commit along with all parent commits. As you can see in the diagram above, the newBranch has detached itself from the master and hence will create a different path.

Use the command below:

**git checkout <branch\_name>**and then

**git commit**



Here, I have created a new branch named “feature” and switched on to the new branch using the command **git checkout** .

One shortcut to the above commands is:

**git checkout -b[ branch\_name]**

This command will create a new branch and checkout the new branch at the same time.

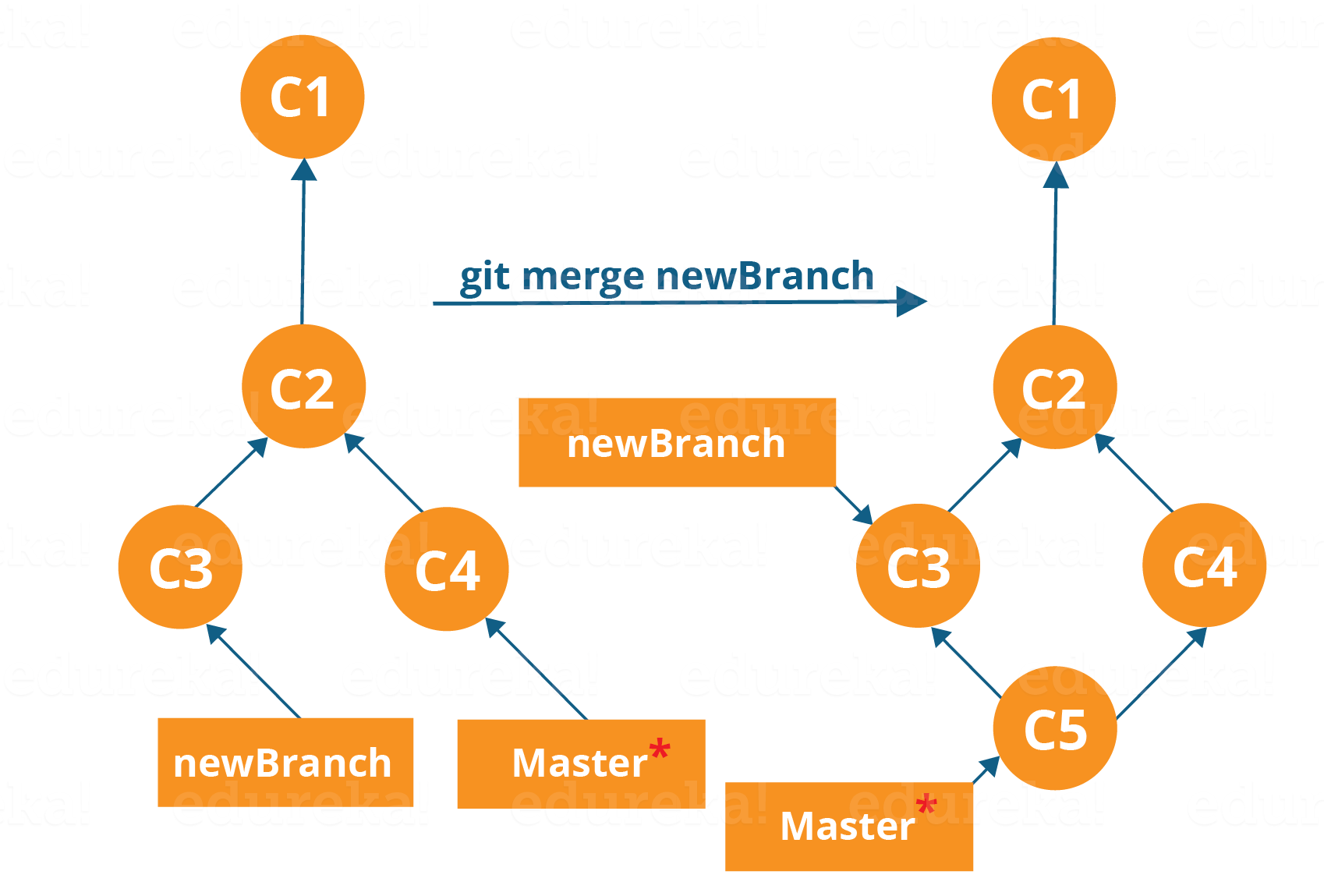
Now while we are in the branch feature, add and commit the text file rami2.txt using the following commands:

**git add rami2.txt**

**git commit –m ”adding rami2.txt”**

**Merging**

Merging is the way to combine the work of different branches together. This will allow us to branch off, develop a new feature, and then combine it back in.



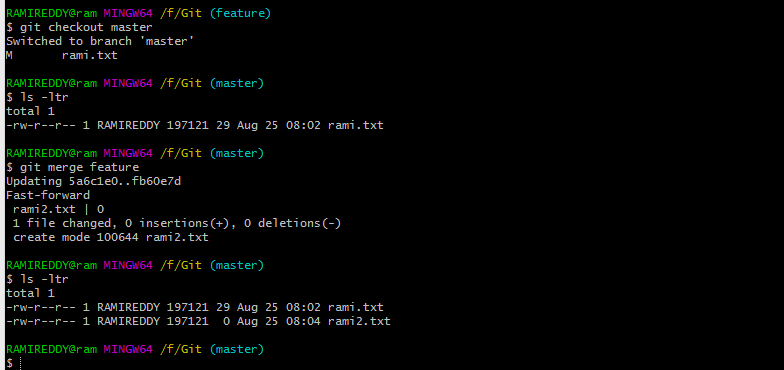
The diagram above shows us two different branches-> newBranch and master. Now, when we merge the work of newBranch into master, it creates a new commit which contains all the work of master and newBranch.

Now let us merge the two branches with the command below:

**git merge <branch\_name>**

It is important to know that the branch name in the above command should be the branch you want to merge into the branch you are currently checking out. So, make sure that you are checked out in the destination branch.

Now, let us merge all of the work of the branch **feature** into the **master** branch. For that I will first checkout the master branch with the command **git checkout master** and merge feature with the command **git merge feature**



If you see the above screen shot, i have create one new text file called rami2.txt in branch of “feature” and has been added or merged into the master repository

Merging in Git creates a special commit that has two unique parents.

**How to delete a local branch in GIT?**

git branch –d <branch-name>

git branch –D <branch-name> <force to delete>

**How to delete a branch from remote as well?**

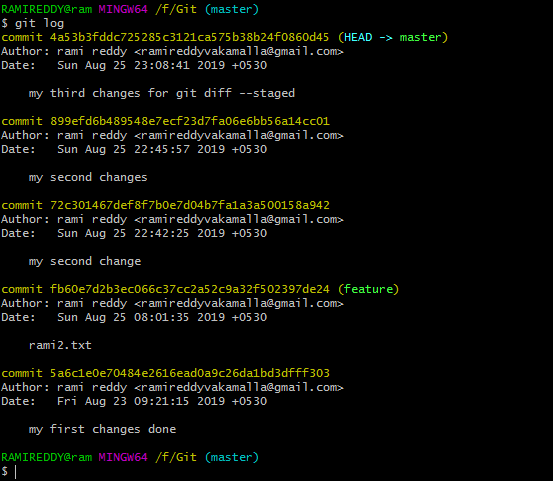
git push origin –delete <branch-name>

**How to see the list of version for the current branch?**

**git log :**

Log command will display all the commits what we did. Or

This command is used list the version history for the current branch.

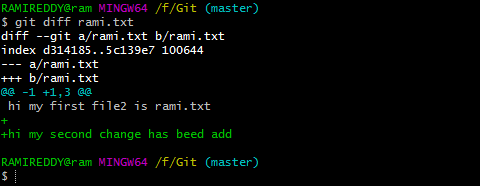


**How to see the modified files which are yet to move to staging env?**

**git diff:**

This command shows the file differences which are not yet staged.

Means: if you are made some changes in working directory and not yet moved to staging env using git add.



**How to see the modified files which are yet to move to local repository?**

**git diff --staged:**

This command shows the differences between the files in the staging area and the latest version present.

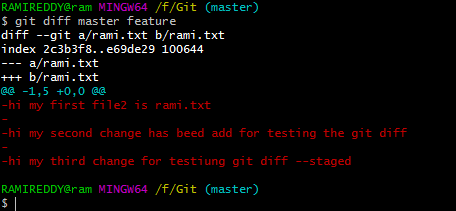
Means: If you are done some changes in working directory and moved to staging env using **git add** but not yet moved to local repository using git commit.



**How to see the differences between two branches?**

**Usage: git diff [first branch] [second branch]**

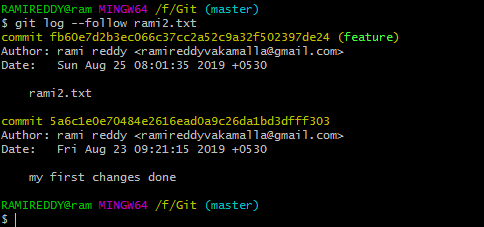
**The command shows difference between two branches mentioned.**



**How to see the list of version history of particular file?**

**git log --follow file-name :**

The command is used to list the version history of the specified file.

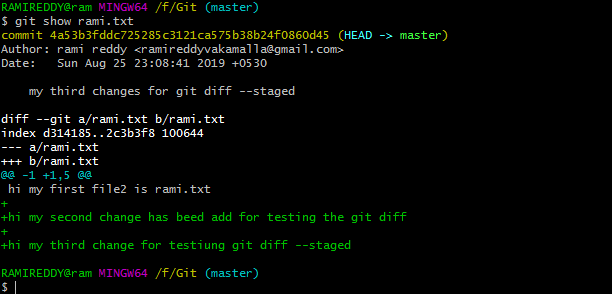


**How to see the recent changes from local repository?**

**git show <file-name>**

The command is used to display the recent changes that we made in the local repository using git commit command.

Note: The below green content represent that we recently added to local repository.

****

**Note: yet to find the below cmds**

**stash**

**Fork**

**Rebase**