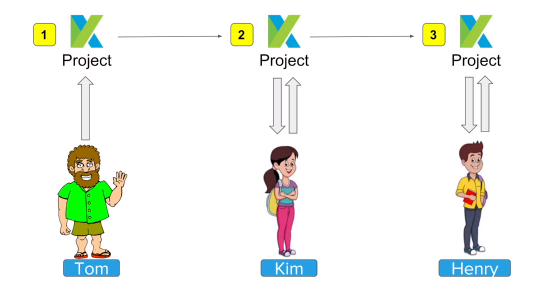
1. **Why to use GIT (version control system)**

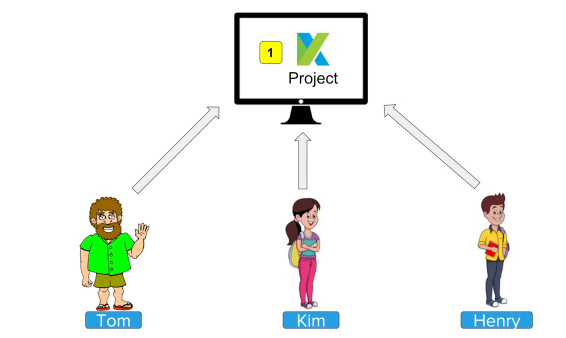
<https://www.youtube.com/watch?v=DqtZUvmPmo4&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG>

Here, we shall discuss why to use version control system like Git. We shall discuss it in the light of a real time example. Suppose, Tom, Kim and Henry are working in a team and they are working in a project. Suppose Tom has created a project and has given to Kim. Let’s say the copy of the Tom’s project is copy1 of the project. Kim then made some changes to it and let’s say it is copy2 of the project. Kim then handed over the project to Henry, who then made changes to it and let’s say it is copy3 of the project.



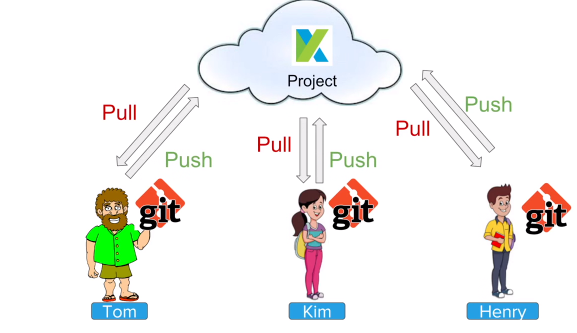
Now, we have three copies of the project. Now, we have the challenge to merge all the copies of the project and make a final copy of the project. One of the options will be, Henry makes his changes and send his project to Kim. Kim merges her change and sends it to Tom. Then Tom merges his changes, reviews and maintains the final copy of the project.

It’s a very manual approach, time consuming, error prone and highly inefficient. Now, what to do now? They can use a common machine, which contains the single copy of the project. This machine can be a remote or on-premises machine. Now, every member will then login to the machine to work on the project.



But, this approach has few challenges. Only one person can work at a time. The team still can’t maintain any change history. Again, an earlier state of the project can’t be reverted back. In the server, multiple copies of the project need to be maintained – one copy per change. Again, this approach is manual, inefficient and time consuming. So, we are back with the same problem. What to do now? We need a solution, which can manage all our changes and version controlling in an automated way.

The solution is using a version control system (VCS) like git. Using git, the team can keep the project in a remote repository like GitHub. Then every member needs to install git in his machine. After that each member of the team pulls a copy of the project from the remote repository and keeps it in his local machine. Therefore, each local machine will have a copy of the project. Each member of the team then works on the local copy of the project and whenever he wants he can push it back to remote repository. Git takes care of all the changes and version controlling.



Let’s see in more details what will be steps,

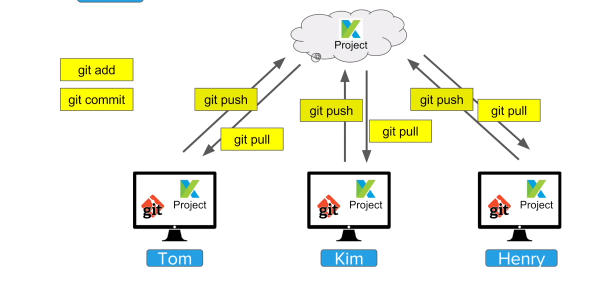
Step-1: Each member installs git in his machine.

Step-2: Anyone can put a copy of the project on remote repository like GitHub or BitBucket.

Step-3: Everyone will then copy the project from the remote repository to the local machine.

Step-4: Everyone can now work on the local copy. While a developer in the team works on the local copy of the projects, there is no need that he needs to be connected to the remote repository continuously. Even if there is no Internet connection, developers can work on their local copy of the project. Network connection is required only at the time of cloning, pulling and pushing of the project.

Step-5: Anyone can commit and push the changes to the remote repository. Therefore, they will do git add 🡪 git commit 🡪 git push.



This is how Tom, Kim and Henry use version control system and resolve their entire problem.

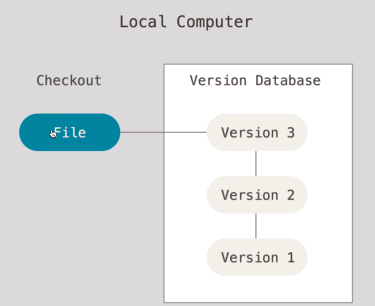
1. **Git and GitHub Beginner Tutorial 1 - Introduction**

<https://www.youtube.com/watch?v=-U-eUHI6euM&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=2>

This section deals with following topics,

1. What is git
2. What is GitHub
3. Is git related to GitHub
4. A simple workflow

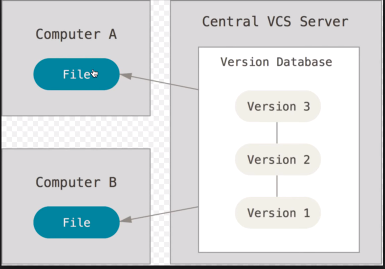
Git is a version control system (VCS). Now, what is version control system? Suppose we have a file, which we want to work on. We need to make change on the file. Further we need to maintain all the changes, made to the file and thus we want to keep track all the changes as far as this file is concerned. In other words we need to maintain all the versions of the file, where each change to the file results in a new version of the file. To achieve this we need to use a version control system. The version control system will have all the versions of the file. A version control system maintains a version database, which keeps track all the versions of the file. Therefore, we can go back to any version of the file and find what changes were made for that version of the file to come into existence. Thus the file is controlled by the version control system.



Version control system is immensely helpful for a software development project, where the code is updated multiple times during its development lifecycle. Codes are kept in files and folders. Whenever code is altered it is the concerned files and folders, which are modified. These files and folders, which contain the code of the project can be tracked using version control system. All the versions of the project are kept in the version database of the version control system. We can easily return back to any prior version.

Git is version control system, which maintains a version database. This version database of git maintains all the version of the project. Therefore, git keeps track all the versions of file and folders with regard to a project.

Version control system makes it possible to collaborate among members of a team. For this a central VCS server is necessary. This central VCS server should be accessed by all the members of the team. Suppose two users in two systems: system A and system B are supposed to edit a file. The file should be kept in the central VCS server and both the users should access it. Suppose the file is checked out by the user in system A and then the user on system A make some modification. After modification he can push the file to the central VCS server. In the same way user on system B can make changes to the same file. Thus, collaboration between two users on system A and B takes place through the central VCS server.

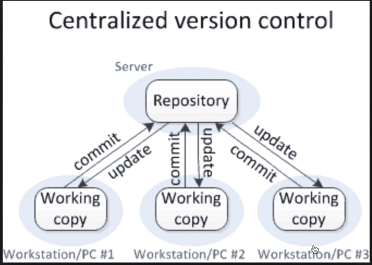


Git also facilitates collaboration among team members as described above. Git can work with various centralized VCS server like GitHub and BitBucket in order to achieve this.

There are two types of version control system (VCS) and they are centralized (CCVS) and distributed (DCVS). In the following we shall discuss about them.

**Centralized Version Control (CCVS):**

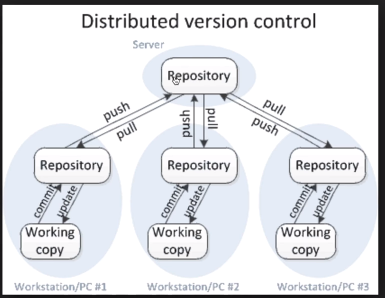
In this version of VCS, the repository is placed in a server. Different people work on this repository, sitting in different workstations or PCs. This workstations or PCs are connected to the server through network. Each developer makes a working copy of the project in his workstation or PC. After that he starts work on it. After he finishes his work he needs to commit the changes to the central copy of the project, which is kept in the repository in the central VCS server.



Here, all the users, who are collaborating, should be online with the repository, kept in the centralized VCS server. This is needed because users, sitting in workstations or PCs, need to do update and commit operation very frequently. Because of this it will not be possible for the developers to collaborate if the central VCS server goes down or connectivity between workstation and central VCS server is lost. Also, repository, kept in the central VCS server will be unavailable if it goes down and no failover mechanism is in place. To overcome these shortcomings distributed version control system (DVCS) came into picture. In the following DVCS has been discussed.

**Distributed Version Control System (DVCS):**

In DVCS, every developer will have a local copy of the repository. Along with it there will be a working copy on which the developer will work. Therefore, developer will first pull the repository, kept in the centralized VCS server into its local system. After that he will create a working copy of it. After that he will work on the working copy. After all the changes are made by him, he will commit them to the local repository. Finally, the developer will push the repository to the server if all the changes he made are fine and satisfactory. Since, he has the local copy of the repository there is no need for him to be connected to central VCS server all the time. He can work offline in his workstation or PC. He needs to connect to central VCS server occasionally – during pulling and pushing the project. Therefore, even if central VCS server goes down work of the developers does not get affected. Moreover, if the central VCS server gets crashed, developer can pull the project from a fellow worker’s workstation, because same copy of the project resides on the workstations or PCs of multiple developers.



Git is open source and free to use.

GitHub is a website to upload our repositories online. GitHub provides several advantages and they are as follows,

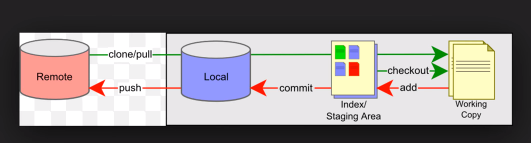
It provides backup. Therefore, our repository is backed up in the cloud.

It provides visual interface to our repository. It helps us to manage our repository easily.

It makes collaboration easier. In GitHub we can create our repository and then make it private or public. We can then give access to this repository to a group of users. Those users will then install git in their local systems. Git on the local system will communicate with GitHub and thus collaboration among several developers is made possible.

Is Git is equal to GitHub. The answer is no. Git is a distributed version control system, whereas GitHub is a website, where we can place our repository. We can use Git with several online repository management systems like GitHub, BitBucket – but in most of the cases Git and GitHub work together. Therefore, Git and GitHub are two different entities.

Let’s see a very simple workflow of Git. It has been demonstrated diagrammatically in the following,



As per above diagram, there is a remote VCS server and we have a local system. In the remote server repository for the project resides. We can do clone or pull to fetch the project repository from the remote server to our local system. Now, we can check out any branch for us, which will be the working copy for us. Now, we can work on the work copy in the offline. After doing all the desired changes to the working copy we need to execute add command in order to place it in index or staging area. After that we need to execute commit command to place the work in local repository. Till not we have pushed our changes to the remote repository, residing in the remote server, which is GitHub in this case. Once we execute the push command the copy of the project, located in the repository in the local server will be pushed to the repository in the remote server and all the changes, made by us will be merged to the remote copy.

1. **Git and GitHub Beginner Tutorial 3 - Getting started - Install Git windows**

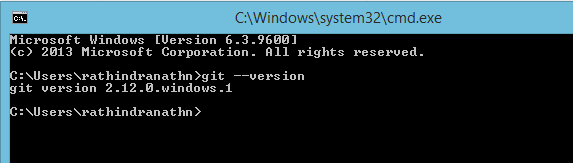
<https://www.youtube.com/watch?v=sBTAkHOxvOk&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=4>

Here we shall discuss the following:

1. How to install Git on Windows
2. Adding project to Git for tracking
3. Git commands
4. Pushing project to remote repository

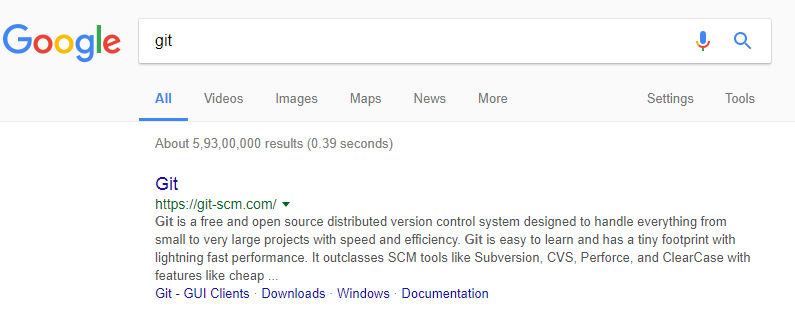
**Installing Git on Windows:**

**Step-1:** Check whether Git is already installed or not in the system. For that we need to go to Windows command prompt and execute following command “git –version”. If Git is installed, command output shows the Git program along with its version.



Otherwise no output is given or command not found error is provided. In that case we need to install Git using following steps.

**Step-2:** Download Git. Go to google search and type “git”. The very first site will be the official site for downloading git as shown below,

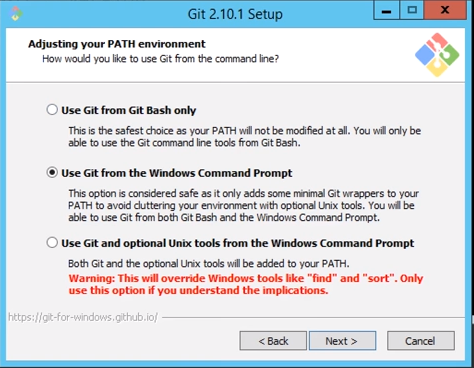


Go to the above link. We can see the download option for git for Windows.



By clicking the download button we can download the latest version of git for Windows system.

**Step-3:** An .esxi file will be downloaded in the above. Now, we need to install it by clicking on it. We can click “Next” on all the default setting. One of the installation steps will be the following,



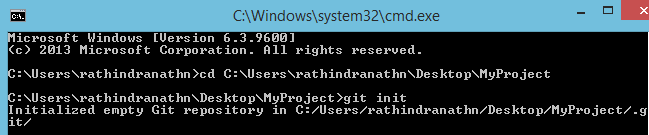
Above window comes by default and if we continue it by clicking on “Next”, we can run git from Windows command line as well as git bash. It is advisable to select the default. At the end we need to click on “Install” and “Finish” on the respective windows.

**Adding project to git for tracking:**

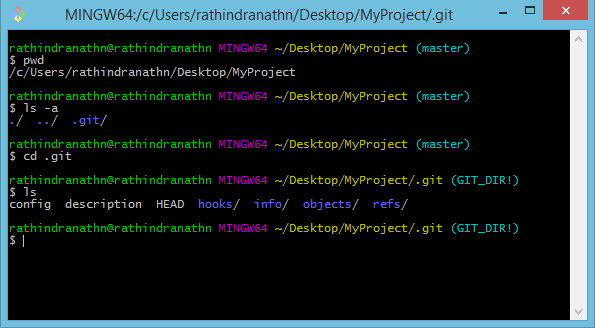
In a very simple term a project is a folder, which contains all the source code and related documents. Let’s create a folder “MyProject” in the desktop. Now, we shall make folder our project folder or project work space. For that, first we need to enter the folder and then invoke git bash or we can invoke Windows command prompt and then change current folder to our project folder. After that in the command prompt we need to execute the command “git init”. To invoke the Windows command line we need to follow the following steps:

Go to the project folder -> Press Windows+R -> Type “cmd” and then Enter

To invoke bit bash we need to first go to the project folder and then do right click and then click on git bash.

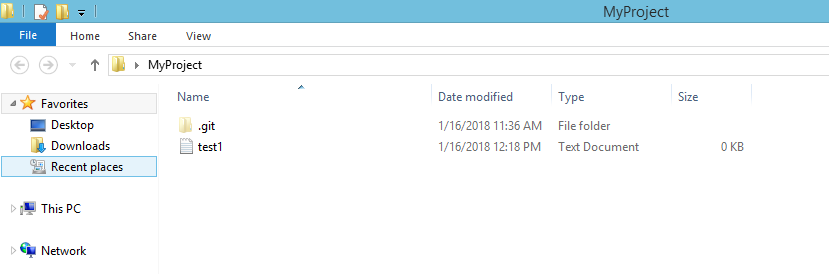


“git init” command generates hidden folder .git in the current project directory and thus the folder gets initialized for tracking by git. .git is the heart if git, which tracks all the changes, made to files and folders in the project directory. The folder .git contains several files and folders as shown below,

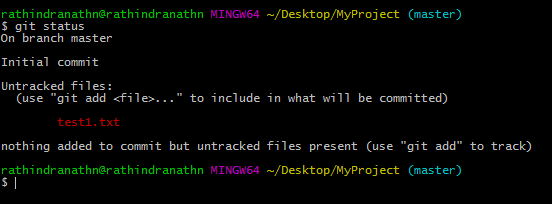


May be .git will not visible in the work space (project folder). To make it visible we need to follow the following steps, open the project folder -> click on “View” -> Check the box for “Hidden items”

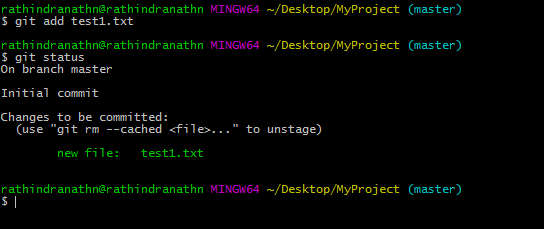
Now we shall create a text file in “MyProject” folder.



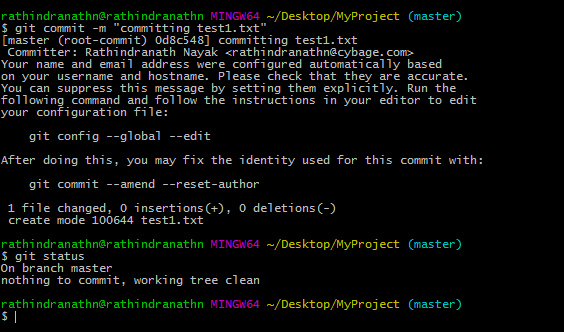
Let’s check the status by executing “git status” command in bit bash,



The command output tells that test1.txt file is untracked and we are advised to execute “git add <file>” command to place it in index or staging area.

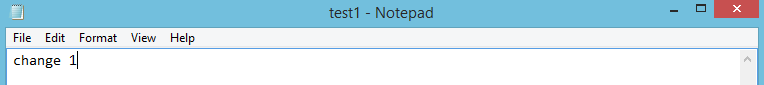


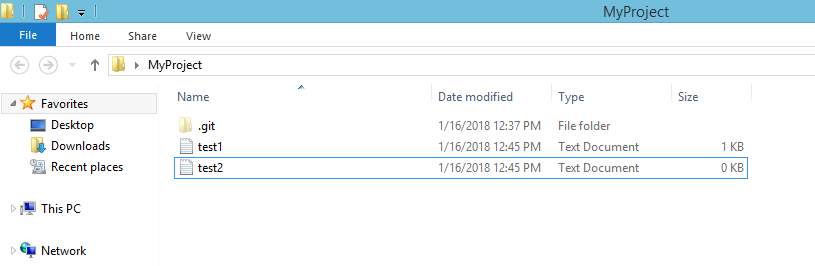
From red, the file test1.txt has turned to green and now it is a candidate for commit. It is now in index or staging area. We shall now commit the file and “git commit ….” is the command for that. During commit we must mention a comment using –m option.



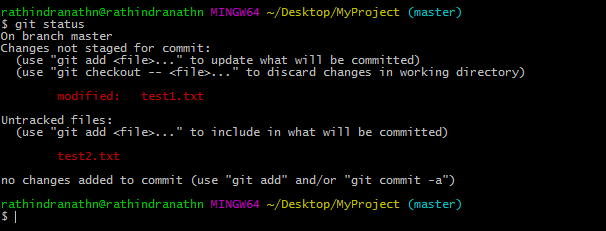
In the above we have committed the file test1.txt and “git status” tells there is nothing left in the staging area to be committed.

Let us make change to the file test1.txt and also create a new file test2.txt.

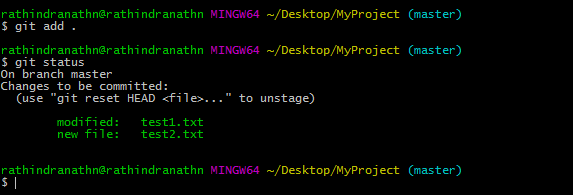




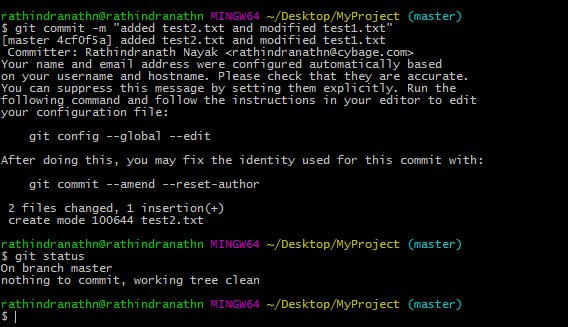
Let’s now check the status,



Above status output tells that we have modified the file test1.txt and we have created a new file test2.txt. Now, we need to move them to the staging area so that both the files become ready to be committed. Let’s add them to the staging or index area and then check the status,



We have used dot or period (.) in the command line of “git add …” to add all the modified and newly created files to the index or staging area. After “git add …” both the files turn green and they are ready to be committed. Let’s commit them and then check the status,



Now working tree is clear and there is nothing to commit.

So far we have used following git command,

* git init
* git status
* git add
* git commit –m “comment ….”

Now we shall push all the changes to remote repository in GitHub. GitHub is a website, where we can keep our project. Whatever we keep in GitHub is backed up automatically and it is impossible that we lose our data, kept in GitHub.

Before pushing our project there we need to create our account in <https://github.com/> and after that we need to login there.

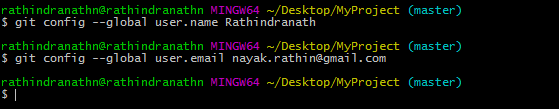
Before pushing our changes from local repository to the remote repository in GitHub, we need to set e-mail and name of the user. In that case every push carries the user name, which helps us to track later who pushed what and when. Moreover, during every push name and e-mail of the user is not asked.

Following is the command syntax for setting the name and e-mail of the user,

git config –global user.email <e-mail of the user as per his github account>

git config –global user.name <name of the user as per his github account>

After initializing our work space so that it can be tracked by git, the first thing we need to do executing above two commands. Let’s execute above to commands in git bash,



One important point - we have already created an account in GitHub with user name as “Rathindranath” and e-mail as [nayak.rathin@gmail.com](mailto:nayak.rathin@gmail.com).

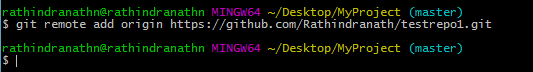
We have created a repository called testrepo1 in GitHub. Now we shall push our changes from our local repository to remote repository. But before that we need to connect our local project folder or work space to our newly created remote repository in GitHub. For that we need to execute following command in git bash in our desktop,

git remote add origin <URL of the git repository>

The URL for our git repository is https://github.com/Rathindranath/testrepo1.git

Therefore the command for pushing our changes to remote repository in GitHub will be,

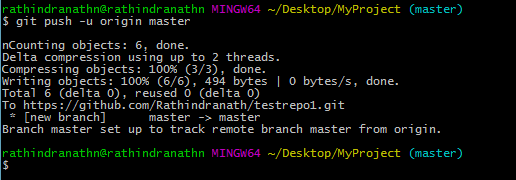
git remote add origin <https://github.com/Rathindranath/testrepo1.git>



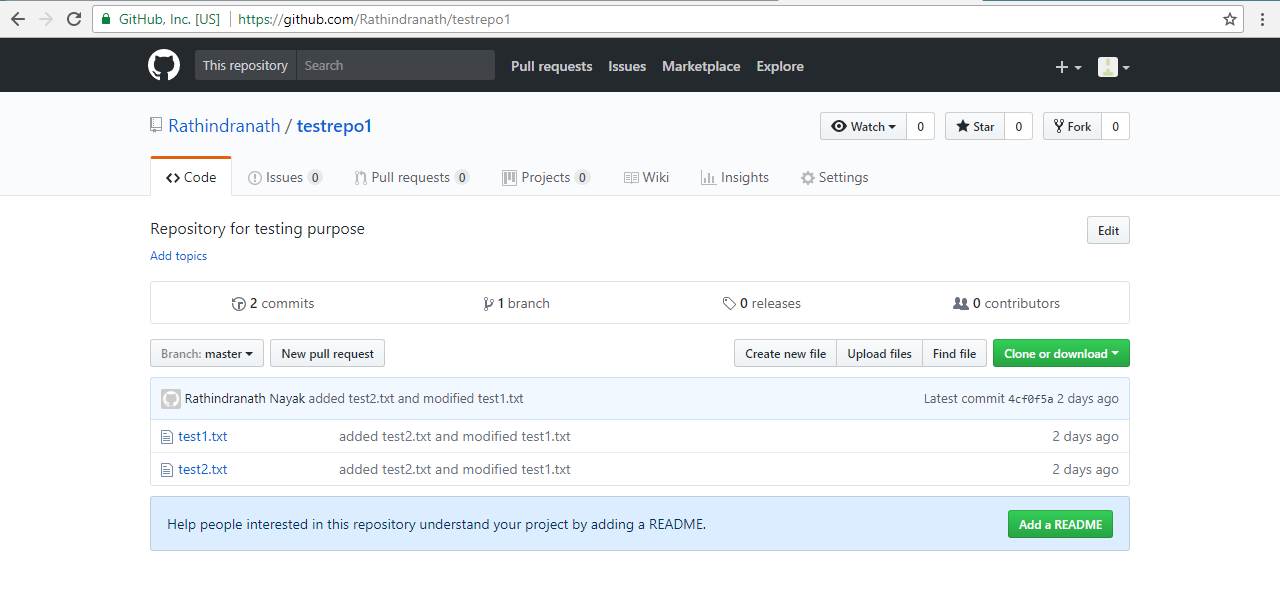
Now we need to execute following command,

git push -u origin master

Above command will push our master branch where all our changes have taken place to our newly created remote repository in GitHub.

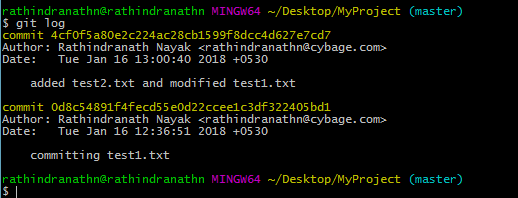


Let’ go to GitHub and check whether contents and change history have been pushed to there or not.



Now, two files test1 and test2 have appeared in our remote repo testrepo2 in GitHub. Those two file, we created and also edited later in our local repository.

There is another command called “git log”, which displays the commit history. Following screen shot shows this,



Above command output shows that we first created a new file test1.txt and then committed it. After that we created a new file test2.txt and also edited existing file test1.txt and after that we committed both of them.

1. **Git and GitHub Beginner Tutorial 5 - Branching and Merging**

<https://www.youtube.com/watch?v=GZILYABgAoo&index=6&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG>

This section we shall discuss following topics,

1. What is branch?
2. How to create branch?
3. How to checkout branch?
4. How to merge branch to master?
5. How to delete branch (remote and local)?

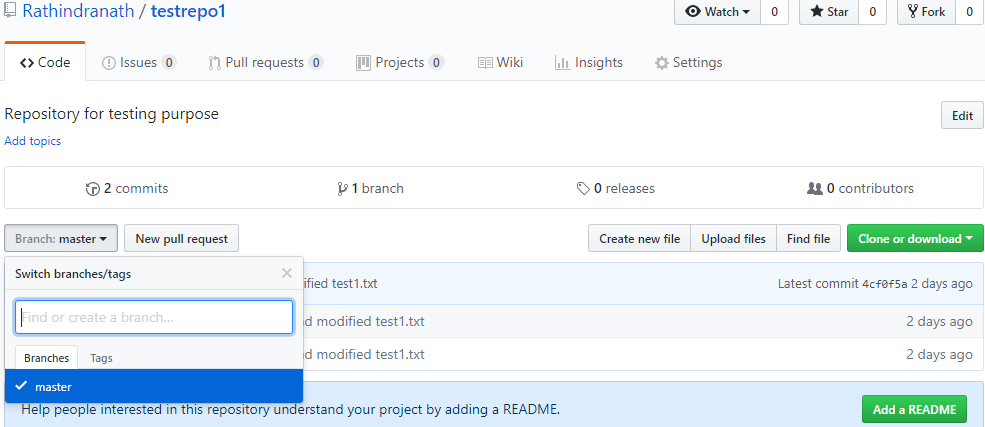
**What is branch?**

When we create a folder for using it as work space for our project, first thing we do is that we execute the “git init” command. This command creates .git directory in the project folder, which keeps track all the changes taking place in the project folder or work space. Along with it one brunch is created, which works as main branch for the project and it contains all the files, which are deployable in the production. This branch is called master branch.

Suppose one developer wants to make changes to the code kept in the master branch. Since it is deployable in the production, nobody should directly touch such code, kept in the master branch and makes the change. To facilitate developer to edit existing code without hampering the code, deployed in the production, git has come with the concept branch. So, he will create a branch out of the master branch. The branch, created from main branch contains all the files and their contents, which are there in the master branch. Developer then switches to the new branch and start working on the contents, kept there. Switching to an existing branch to a new one is called check out to the new branch. After making all the intended changes, the changes are validated and finally the branch is merged to the master branch. In this way through merging all the changes, made by the developer are added to the content, which are already there in the master branch. Finally, the developer can delete the branch, he created.

**How to create branch?**

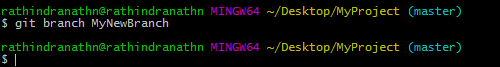
We have already created a new repository called testrepo1 in the GitHub. Screen shot of the repository and its content have been shown in the above screen shot. It also shows that the repository has only one branch and it is master branch. Let’s look at the following. It shows that currently testrepo1 has one branch and it master branch.



Now, we shall create a new branch out of the master branch in our project directory or work space with the help of git. Following are the steps,

**Step-1:** Create branch. Following is the command syntax,

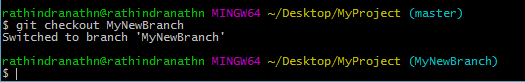
git branch <branch name>



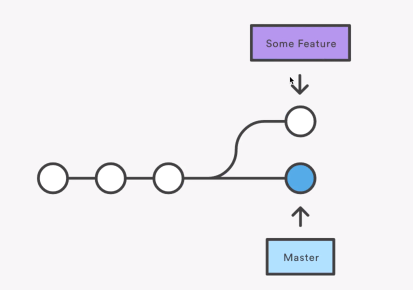
Above command creates a new “MyNewBranch” out of the master branch.

**Step-2:** Check out to the new branch. Still we are in master branch. We need to check out to the new branch in order to work on it. Following is the command for that,

git checkout <branch name>

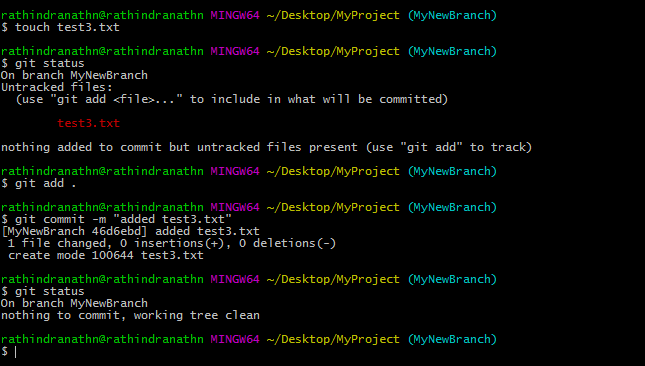


Now, whatever change we do it goes to the current branch, which is the new branch we created. Following figure shows branching,



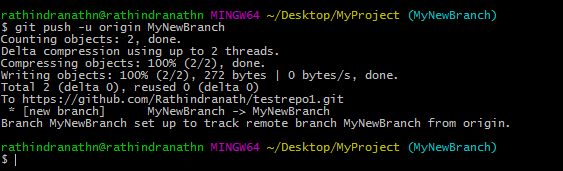
Developer needs to add new feature to the existing code. So he checks out a new branch and makes all the intended changes. After all the changes are made and validated, the branch is merged to the master branch.

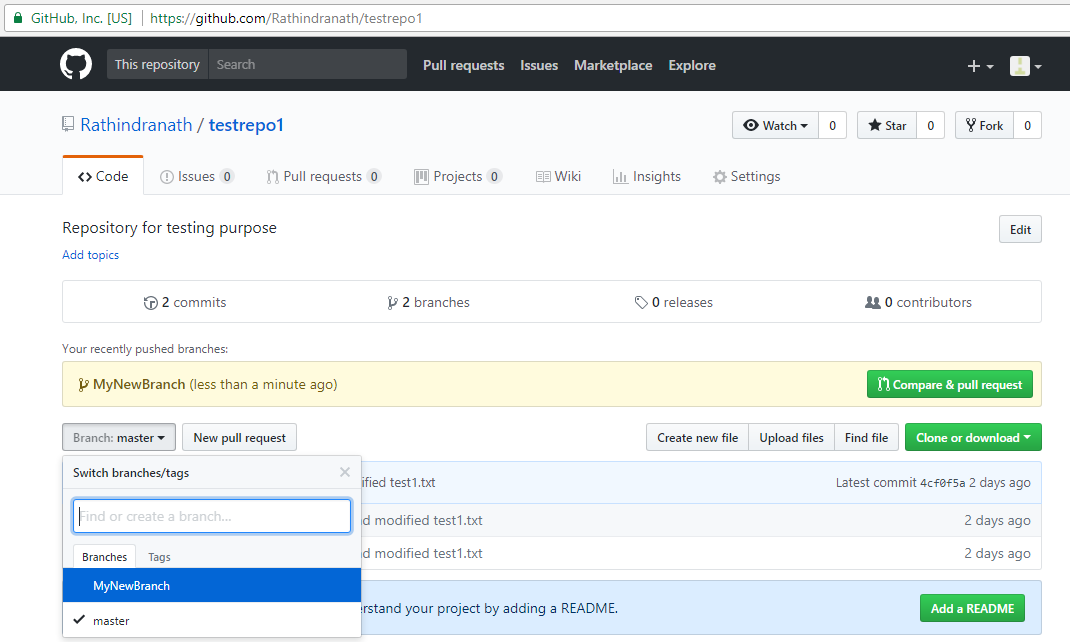
In GitHub till we have one branch, which is the main branch. Let’s make few changes to in the project folder “MyProject”. All the changes we shall do in the new branch “MyNewBranch”.



Now, we shall push the change in the remote repository in GitHub. We shall do that staying in the branch “MyNewBranch”. But there is no such branch in the repository (testrepo1) in GitHub. Following command will push the branch “MyNewBranch” along with the change in GitHub,

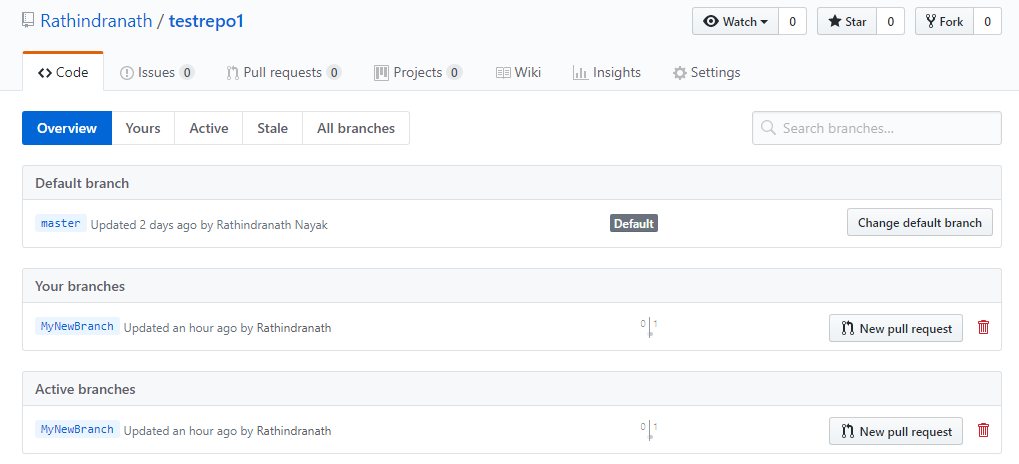
git push -u origin MyNewBranch



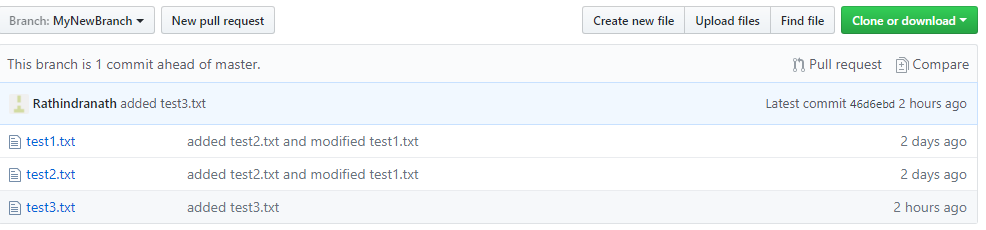


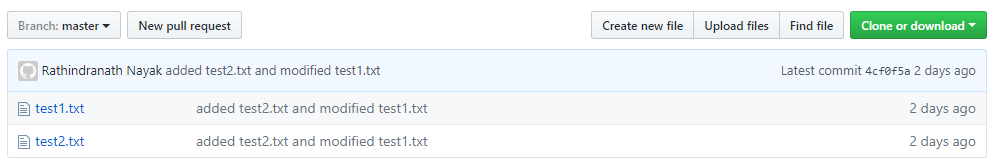
A new branch called “MyNewBranch” has appeared in “testrepo1” in GitHub. Now, the repository has two branches – master and MyNewBranch.

If we click on “2 branches”, it shows the details about them and when they got updated.



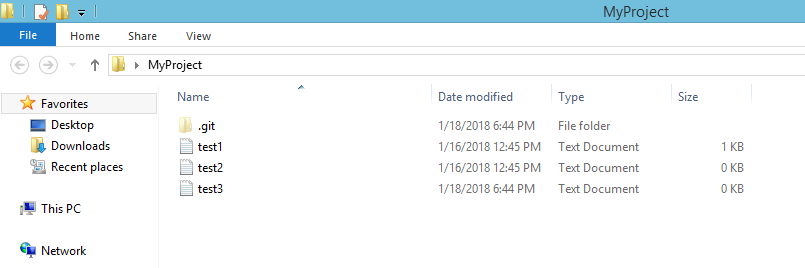
Let’s us click on “MyNewBranch” and “master” respectively. We can do that by selecting the branch in branch drop down.



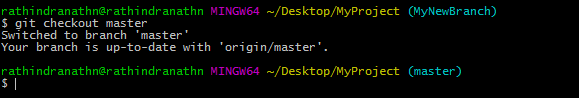


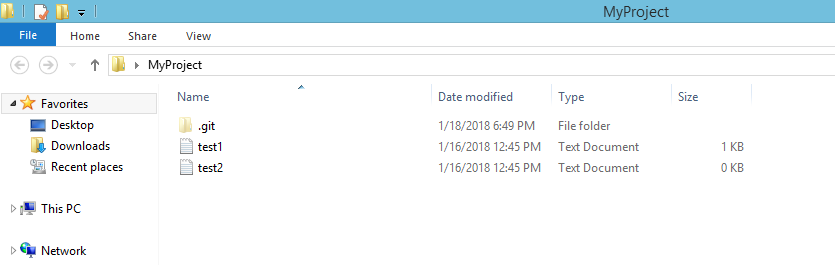
In the master branch there are two files and they are test1.txt and test2.txt. But in MyNewBranch branch there are three files and they are test1.txt, test2.txt and test3.txt.

Same is reflected in our project directory in the local system. Currently we are in MyNewBranch. Let’s look at our project folder MyProject,



Now, we shall switch to master branch then look at the MyProject folder,





MyProject folder does not contain test3 file now.

Since, MyNewBranch is not merged to master branch; test3.txt file is not present there. In the following we shall do the branching operation.

**How to merge branch to master?**

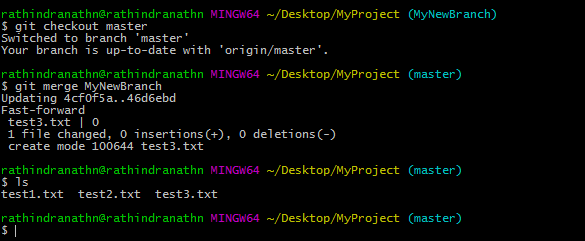
It’s a two steps process.

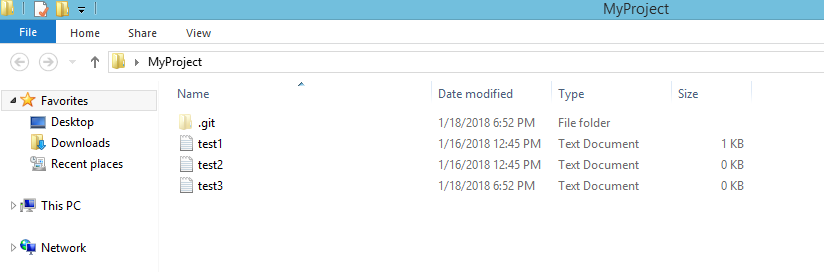
**Step-1:** Checkout master branch. Following is the command,

git checkout master

**Step-2:** Merge the other branch to the master. Following is the command,

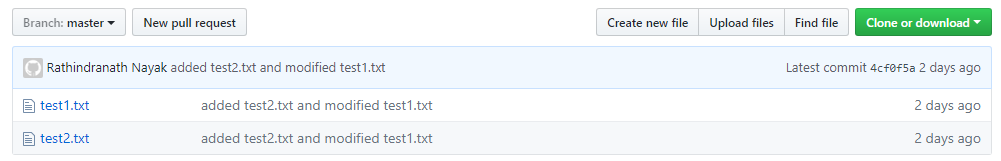
git merge <Branch name>





In the above we have merged MyNewRepo to master branch in local repository. After that test3 file has appeared in master branch.

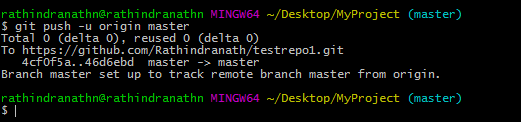
But still test3 file has not moved to master branch in the remote repository in GitHub.

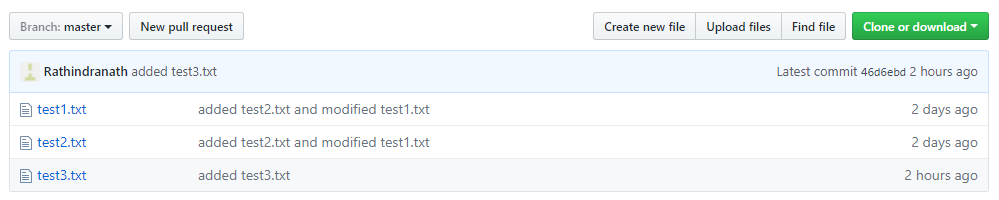


Now, we need to execute following command at git bash prompt in our local system,

git push -u origin master

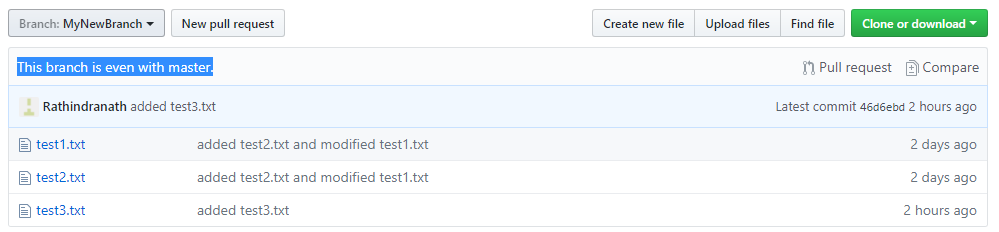
Above command will push the master branch from the local repository in our system to the remote repository in GitHub.





Now, test3 file has appeared in the testrepo1 repository in GitHub.

In GitHub if we switch to the MyNewBranch the message “This branch is even with master” Will be shown.



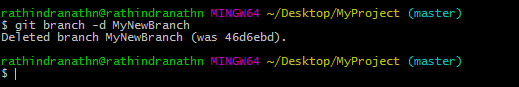
All the changes done in the branch MyNewBranch has been merged to the master branch. And this merge is reflected in local repositories in our system and also in remote repository in GitHub.

**How to delete branch (remote and local)?**

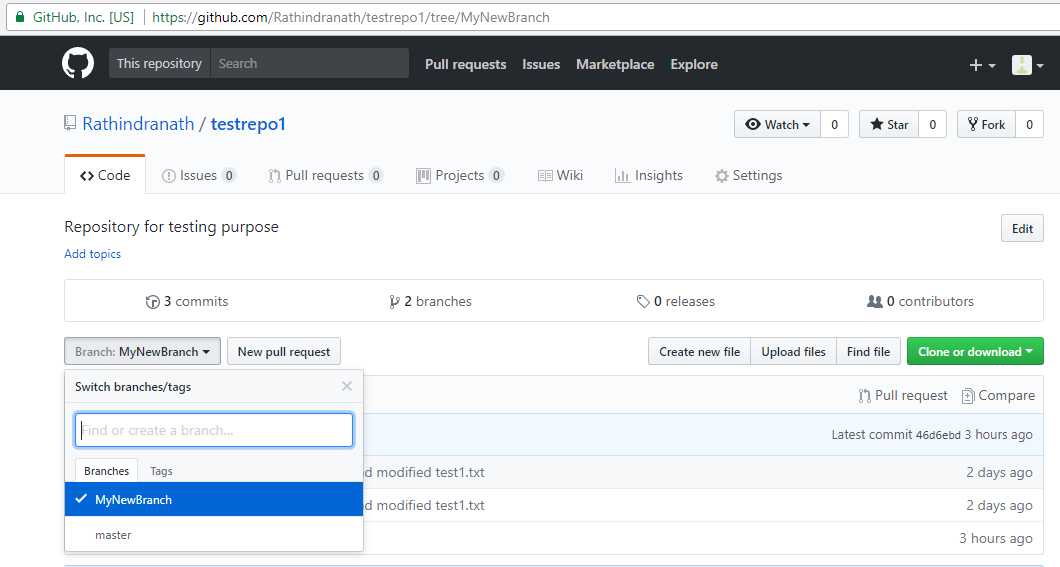
The purpose for creating the new branch MyNewBranch is now fulfilled. We have done the intended change, validated it, merged it locally and then pushed it to the GitHub. We, therefore, no longer need this branch. So we can go for deleting the branch.

To delete a branch we need to execute the following command,

git branch -d <Branch name>



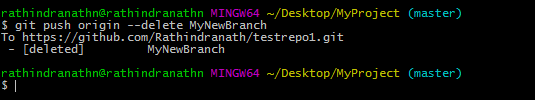
Above screen shot shows that branch MyNewBranch deleted. But it is deleted locally. But if we go to our remote repository testrepo1 in GitHub we can still see the branch MyNewBranch.

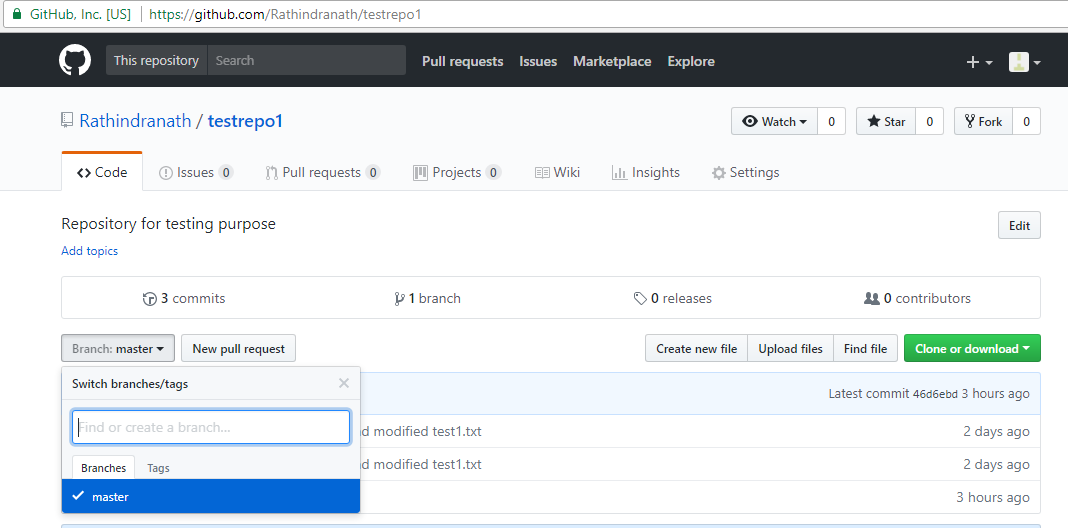


To delete a branch from remote repository in GitHub we need to execute following command,

git push origin --delete <Branch name>

Let’s execute this command in git bash and then check the repository in GitHub.





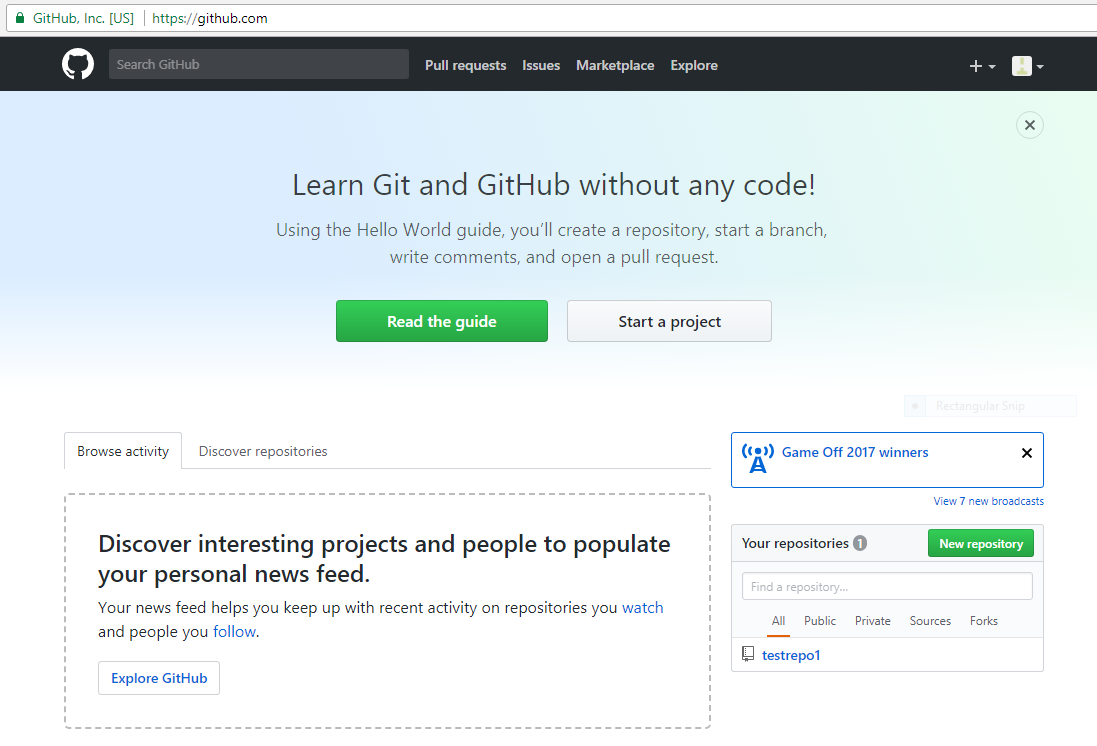
MyNewBranch has been deleted from remote repository also and there is only one branch there – master branch.

1. **Git and GitHub Beginner Tutorial 6 - How to send email from GitHub**

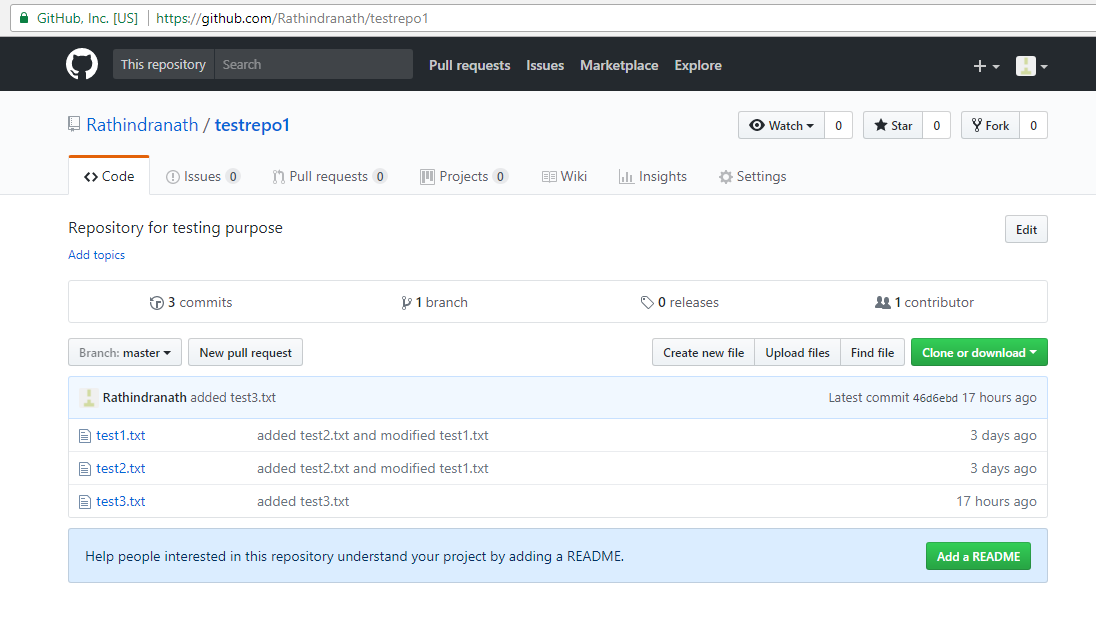
<https://www.youtube.com/watch?v=Ft2LXlaSEfs&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG&index=7>

Here we shall discuss how to trigger notification e-mail from GitHub whenever there is any change/commit in the project. This notification is set for a repository in GitHub. For that first we need to login to our account in GitHub. After that we need to go to our intended repository. After that we need to click in “Settings” in the upper pane of the page. This “Setting” tab is for this repository and it provides us the options to change various settings particularly for this repository. Following is the steps to set e-mail notification for any change made in a particular repository in GitHub,

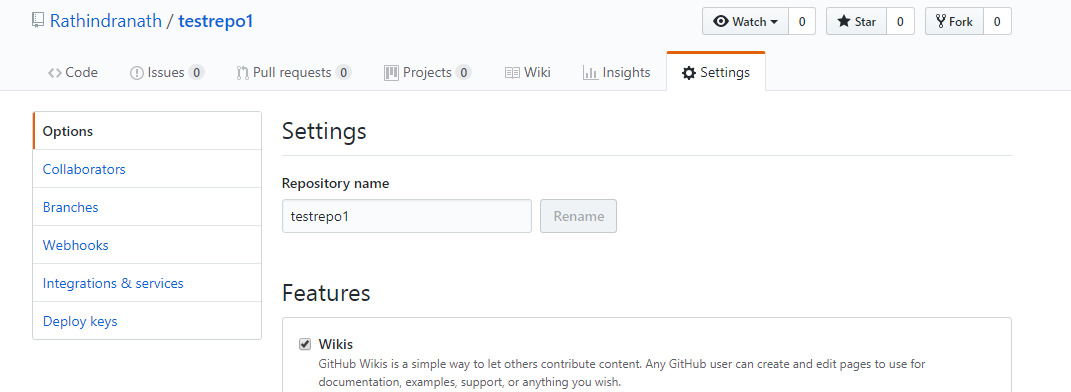
Github -> Repository -> Settings -> Integration & services -> add email



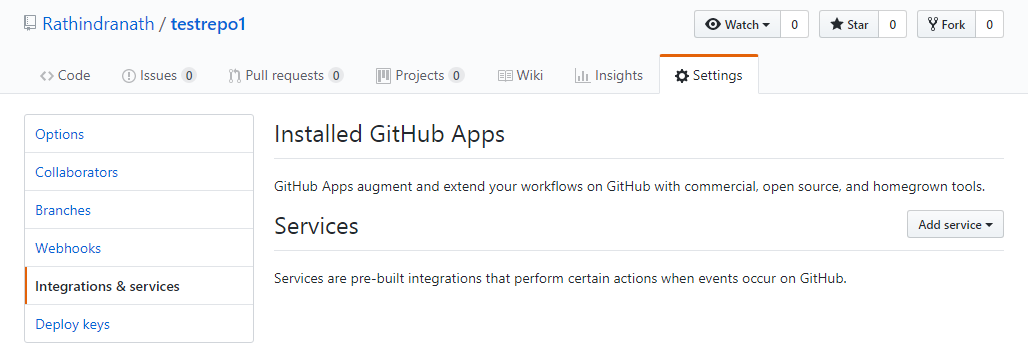
Above is the page we land after we login to our GitHub account. It shows our repositories in this page. Currently we have only one repository and it is “testrepo1”.



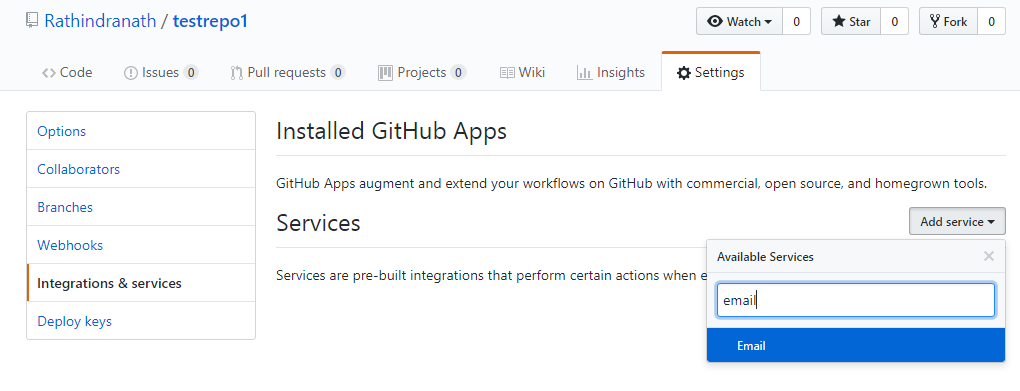
If we click on the repository (testrepo1 in this case), the concerned repository gets open. After that we need to click on “Settings” and following page will appear.



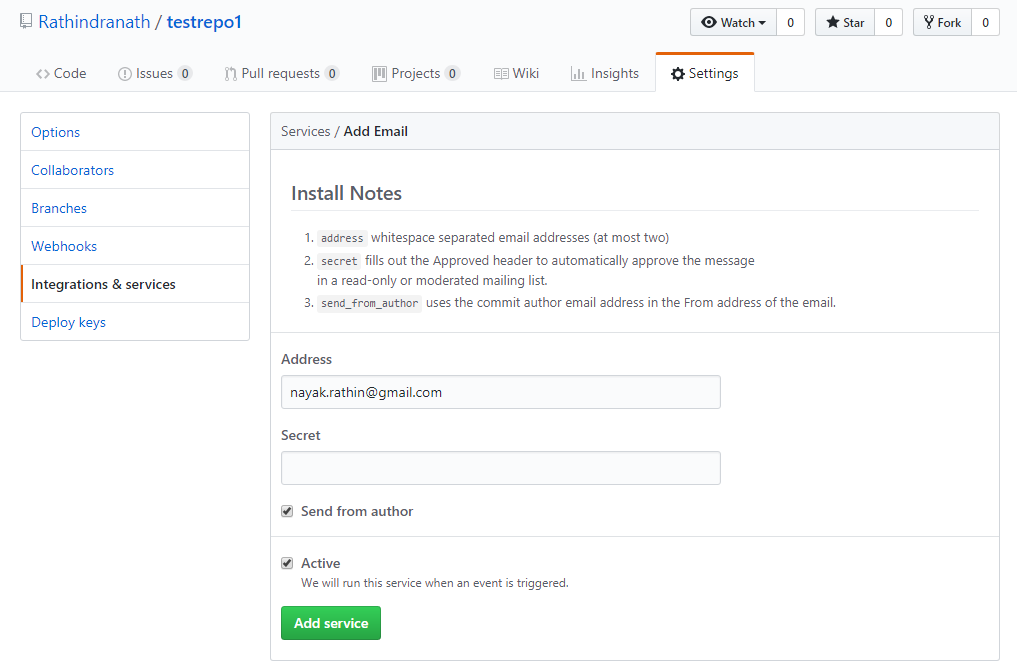
Now, we need to click on “Integrations & service” in the left side of the page. Following page will appear.



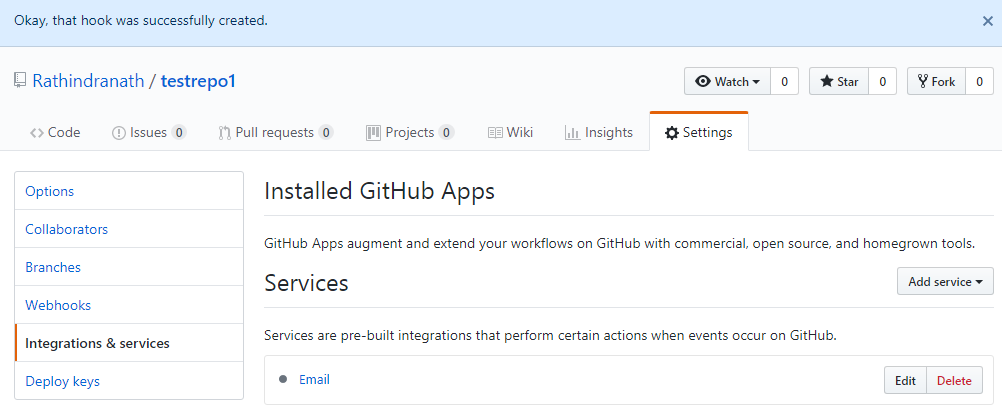
We then need to click on “Add service” and then type the service we need. Following page appears,



We have to click on “Email”. A new page for adding e-mail will appear. We need to mention the e-mail and other fields as shown below,



Now we have to click on “Add service”. E-mail service for that particular repository (testrepo1) will be set as shown in below screen shot,



Now onwards, whenever any change takes place in testrepo1 repository an e-mail notification is sent to mail id, mentioned during setting it. The mail is sent from GitHub and it mentions that a commit happens in the concerned repository. It also mention the commit id, who made the commit and when (date and time) it was made.

1. **Git and GitHub Beginner Tutorial 7 - Git Tags - what, why, when and how**

<https://www.youtube.com/watch?v=govmXpDGLpo&index=8&list=PLhW3qG5bs-L8OlICbNX9u4MZ3rAt5c5GG>

Here, we shall discuss about git tags. Following are the topics to discuss,

1. What are tags/releases?
2. Why should I create TAGs?
3. When to create TAGs?
4. How to manage TAG in Git – create, show, publish, delete?

**What are Git tags?**

Tagging in Git or any other VCS refers to creating specific points in history for our repository/data. Therefore tagging is a process of marking specific commit points in entire history of our repository.

Tagging is done to mark release points. Suppose we have developed a project and we want to mark it as release 1.0. We can create a tag for that.

**Why should I create TAGs?**

We should create TAGs for following two reasons,

1. To mark release points for our code/data
2. To create historic restore points.

**When to create TAGs?**

Following are two occasions when we need to create TAGs,

1. When we want to create a release point for a stable version of our code.
2. When we want to create a historic point for our code/data that we can refer at any future time (to restore our data).

**How to create TAG in Git?**

This is a four step process,

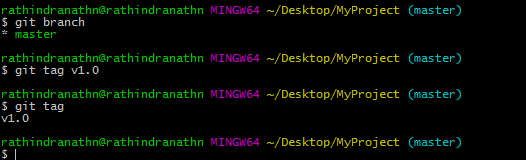
Step-1: Checkout the branch where we want to create the tag,

git checkout <branch name>

Step-2: Create tag with some name,

git tag <tag name>

Example: git tag v1.0

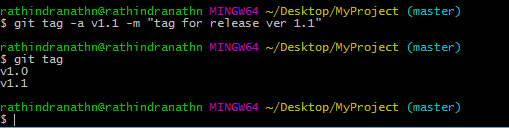


In the above we have created a new tag v1.0 in the master branch. We have also checked using “git tag” command whether the tag has been created.

We can also create an annotated tag by using the following command,

git tag -a <tag name> -m “message….”

E.g. git tag -a v1.0 -m “version of 1 of the project”



In the above we have created annotated tag v1.1. The command “git tag” shows two tags – v1.0, v1.1. In annotated tag we can contain message and also it contains all the information about the tagger. An annotated tag is stored as a complete git object in git repository.

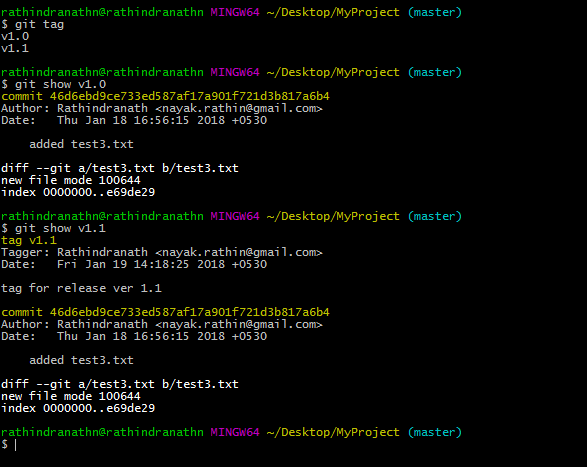
Step-3: Display or show tags,

To display all the tags we need to execute following command,

git tag

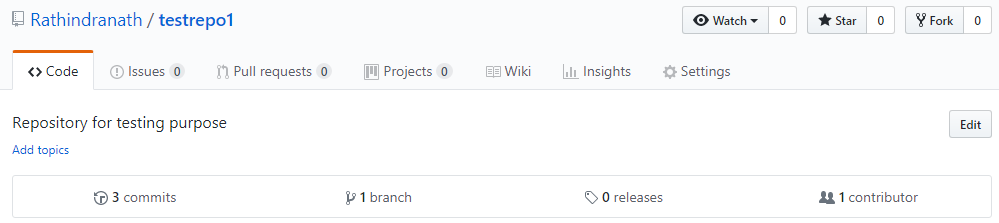
To display the details of a particular tag we need to execute following command,

git show v1.0



Step-4: Push tags to remote

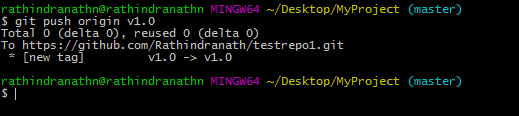
If we go to our remote repository in GitHub we can see that there is no release or tag. The reason is still we Have not pushed the tags to the remote repository in GitHub.



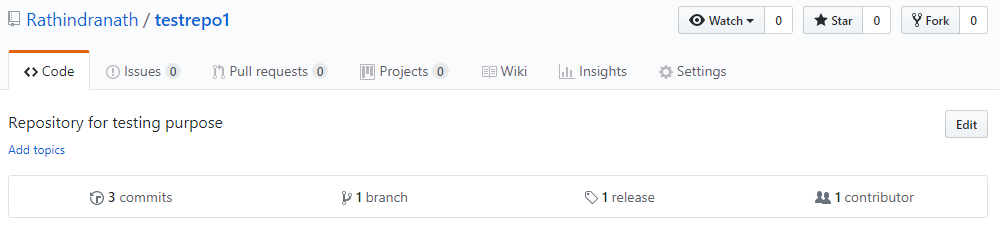
To push tags to remote we need to execute the following command,

git push origin <tag name>

Let’s push the tag ver1.0 to our remote repository testrepo1 in GitHub,



Let’s check the repo in GitHub,

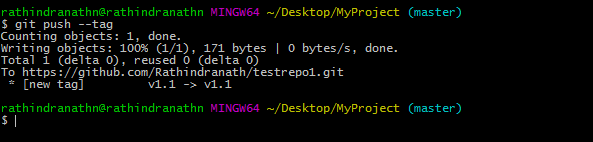


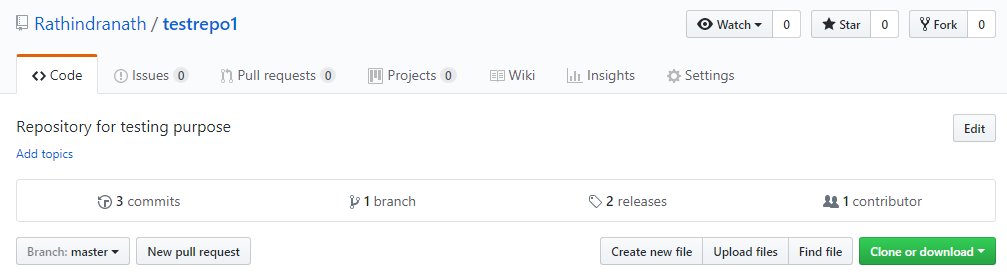
Number of release is showing as 1.

To push all the tags to the remote repo at a time we need to execute one of the following commands,

git push origin --tags

git push --tags





Now, the number of release is showing is 2.

**Deleting tag:**

We delete tag if it is at all required. To delete a tag we need to execute either of the following commands,

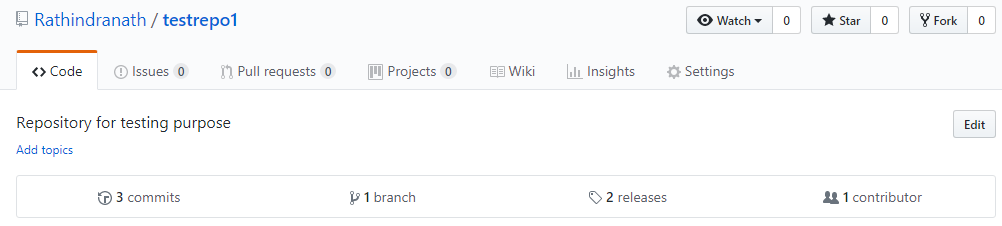
git tag <tag name>

git tag --delete <tag name>

Let’s delete the tags we have created so far,



But still they are not deleted in GitHub. Following screen shot shows this,



Number of releases is still showing as 2.

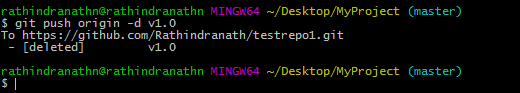
To delete a tag in remote repository in GitHub we need to execute any of the following commands,

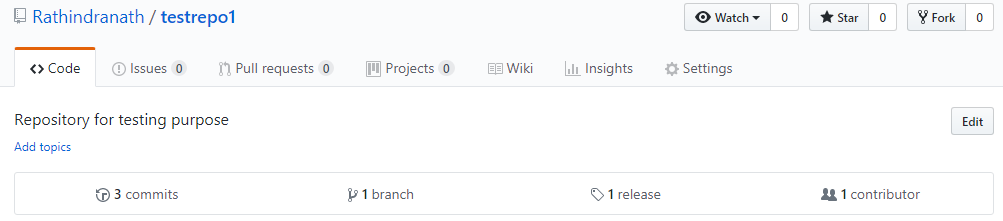
git push origin -d <tag name>

git push origin --delete <tag name>

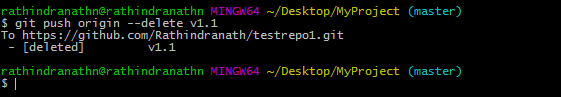
git push origin :<tag name>

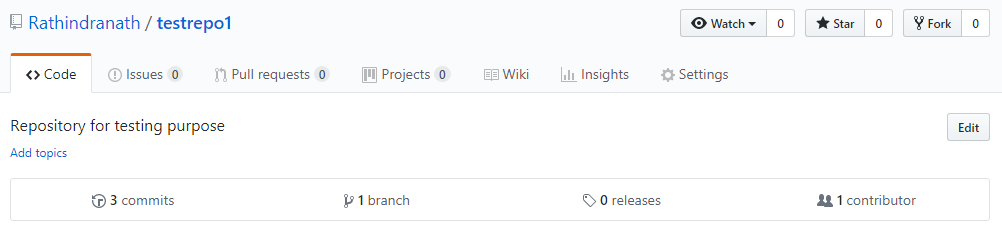
Let’s delete the tag v1.0 from the remote repo in GitHub,





In GitHub one release has been deleted, but still one release is there. Let’s delete that also,





Now, there is no release in the remote repo testrepo1 in GitHub.

We can also delete multiple tags at a time locally or remotely. For that we need to execute following command. The tags are separated by space in the command line.

git tag -d <tag name1> <tag name2> …. (local)

git push origin -d <tag name1> <tag name2> ….. (remote)

**Significance of TAGs:**

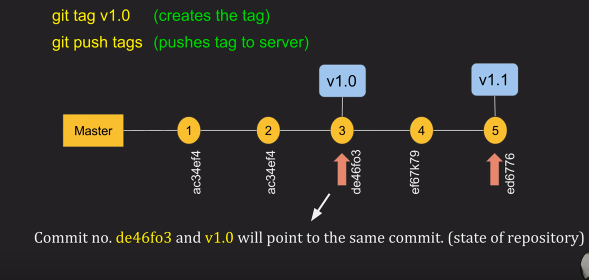
Let’s check out the master branch. After making change we need to commit. Each commit is represented by a circle with a number within it. So we have commit point 1. Now, we shall make change in the master branch again. After doing so, suppose we have committed it. It is represented by commit point 2. Then there is another change and then commit. A new commit point 3 results in. In commit point 3, we have reviewed the project and have found out that it is stable. Now, we want to take a historic point of the repository at commit point 3. So we execute the command,

git tag v1.0

Then we can push the tag to the remote sever,

git push --tags

But still we continue changing the repository. After changing few stuffs we have committed the change and the commit point for that is 4. After that we did commit 5. At this particular point we again want to create a tag. Let’s say it is v1.1. Every commit bears a 40 character long unique checksum. It is a checksum number and it is generated by SHA1 algorithm. Therefore commit checksum for commit point 3 and v1.0 refers to the same commit.



Now, how to checkout TAGs (when required)? We can’t checkout tags in git. Instead, we can create a branch from a tag and checkout the branch. Following single command will do this,

git checkout -b <branch name> <tag name>

For example, if we want to checkout tag v1.0 we need to execute following command,

git checkout -b ReleaseVer1 v1.0

Here we have given the branch name “ReleaseVer1”. Above command will checkout a branch named ReleaseVer1, which will have the same state of the repository that we tagged at v1.0.

Can we create a tag from some past commit?

Yes, it is possible. Suppose we want to go back to a commit point and then create a tag for that commit point. For that we need to execute following command,

git tag <tag name> <reference of commit>

Reference of commit is the 40 character checksum for that particular commit. Suppose we want to tag the commit point whose checksum is 5fcdb03……. Following is the command,

git tag v1.2 5fcdb03

No need to enter full 40 character long checksum, first few characters are fine as long as it is unique in commit history. We can now push the tag to our remote repository in GitHub. For that we need to execute following command,

git push origin v1.2