Version Control System
$git

Prof. B. Thangaraju

Why do we need Version Control System (VCS)?

- If you are a software developer and working in a project team, your main job is to develop code for a specific module in your project. Initially you may develop a code locally in your system. If your project code is getting bigger and bigger then you should maintain the code carefully.

- Suppose you are working on a code and after making many changes you realize that you have really messed up or the current version of your code may have some issues and now you would like to revert to the last good version of your project. How would you do that?

- if you are not maintaining copies of the various versions of your code then you will be in trouble. So how do you revert to the previous working version of your project code?

Version Control System

- In other scenario, if you are working in a project team and develop a module with a group of team members, every time you work on the project, you should know exactly what has already been completed, added, changed and so on. How would you know who made the changes, to which files were changed. Otherwise, you may end up working on something that is already been finished.

- Now, how do you share your recent code update to your team members? One way is that you can compress your code with all the necessary artifacts and send every updates through email then others can start to work with the module. If a large group of developers share the code updates through mail then someone should collate all the updates into the whole project code. This process will become a hectic task and error prone.

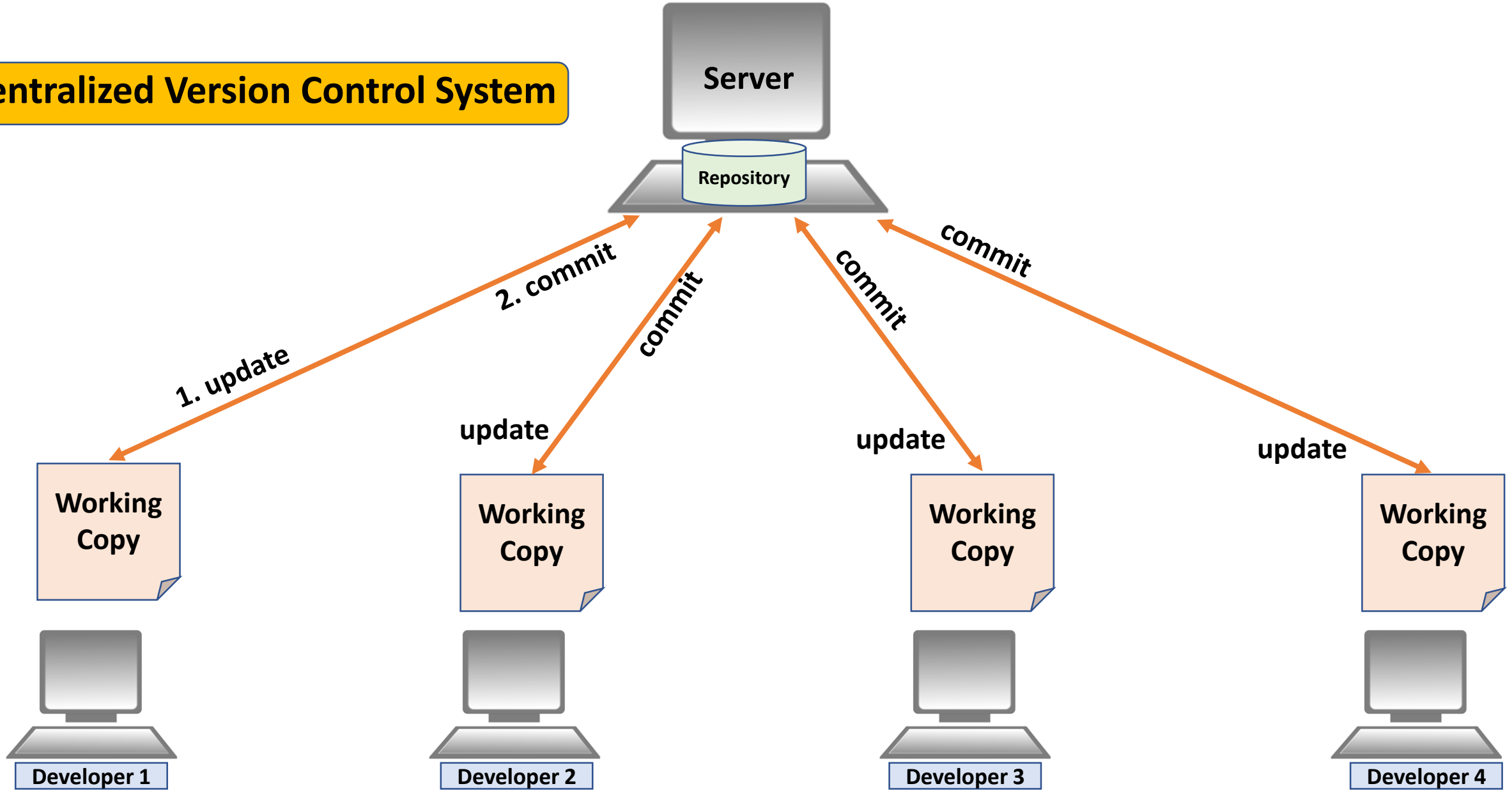
- This is an inefficient way to share the code updates to the team. Do we have any better solution to share the code to others and maintain history of each and every updates in a systematic way? The solution is Version Control System.

Centralized Version Control System

- Version control system is the management of changes to collection of information like documents, project code.
- It can track collaborative changes to a project, so a developer is aware of the recent changes in the project code and he can access the most recent version of the project.
- Also developers can view all the past versions of the code and the difference between them.
- There are two major types of version control model namely
  - 1. Centralized
  - 2. Distributed

The **centralized version control system** is working as a client-server model. Here we have one centralized server and a localized repository filesystem that is accessible by a number of clients. The advantage of this centralized model are simplicity and ease of use. One of the examples of this model is Subversion.

Centralized Version Control System



Centralized Version Control System

- Google doc is an example of centralized version control system because google doc only keeps one file for the word document, and everyone works on that one file. Therefore, all the changes and modifications will be stored and get reflected on that one common file.

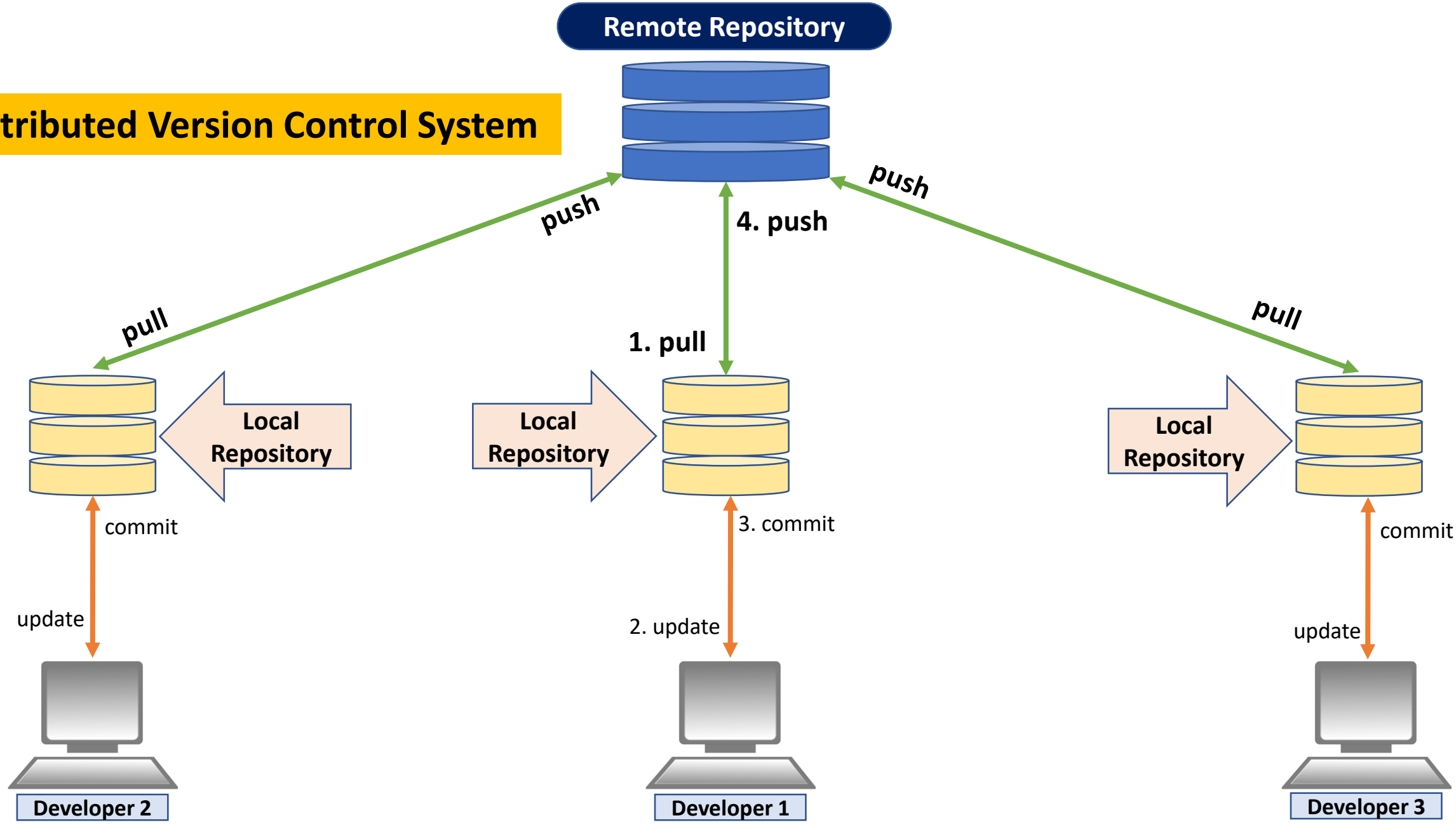
- In Subversion there is only one repository filesystem, usually stored on a network server. Whenever you want to work on your project, you can connect the central file system through network connection. So you cannot do any work on your project without a network connection. If anything happened to the central filesystem you may lose your data and or history of the changes.

- So the disadvantages are the Subversion is completely dependent on the functionality of a single server, which can become a bottleneck and may affect the performance due to heavy network traffic and reliability of backups.

Distributed Version Control System

- Distributed version control systems takes a peer-to-peer approach to version control, as opposed to the client–server approach of centralized systems.
- In the distributed model, all developers have their own local filesystem, and changes between file system are implemented locally on their machines. The advantages of this model are faster access, ability to work offline and does not rely on a single location for backups. Example for this model are GitHub and Mercurial.
- Git has some advantages over other distributed version control systems like Mercurial.
  - git is faster than mercurial for network operations such as downloading and uploading project files to the file server.
  - Git's approach to branches are more powerful than Mercurial.
  - git is more powerful for larger projects.
  - Git is one of the most widely-used popular version control system in use today. For example, teams at Amazon and Microsoft have adopted git as their version control system for many of their projects.
  - Git is Open Source and you can create a public or private file system which can be accessible by your project team.

Distributed Version Control System



Git

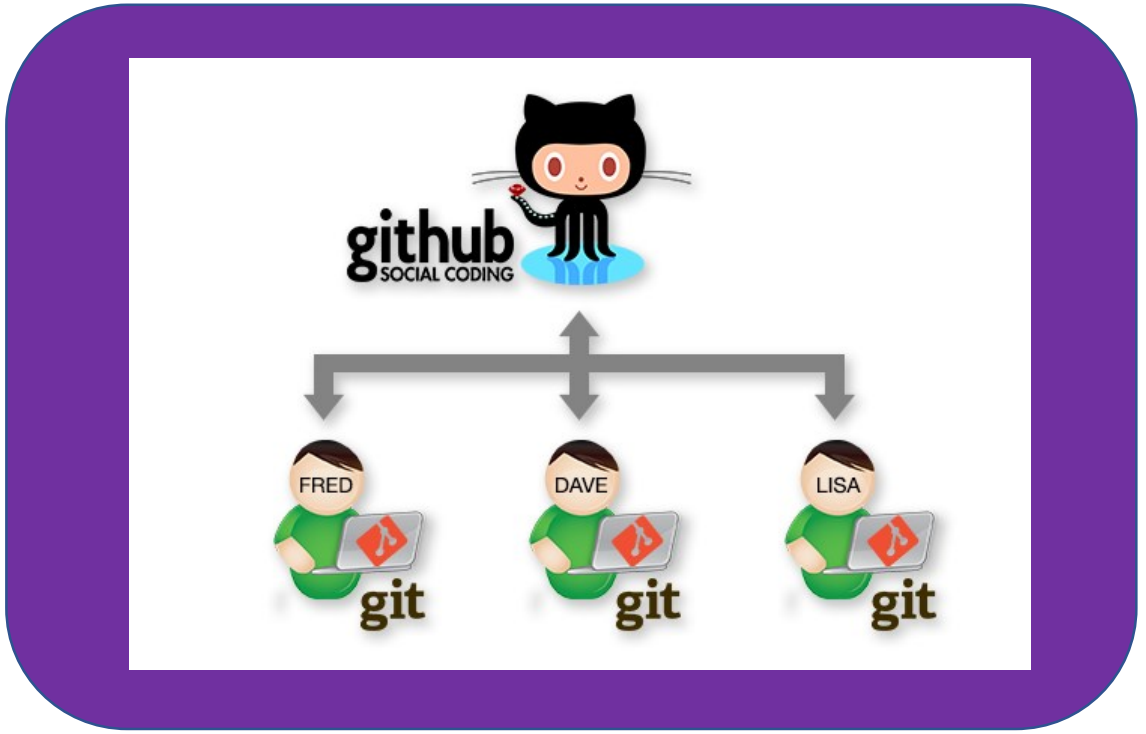
SPE- Thangaraju Sir

- Git is invented by Linus Torvalds.
- The latest Linux kernel-4.13 has 60,538 files and over 24 million lines of code.
- Approximately 3,500 new lines of code are added into the Linux kernel source code every day.
- 1,681 developers are involved from 225 companies. New version of the Linux kernel is released in every quarter.
- To maintain such length code which is being worked upon by thousands of developers there was a need of a tool which could manage such a length code well and make collaborations between developers across different places and companies efficient and easy.
- Git is used to maintain Linux kernel. Git makes collaboration as quick and painless as possible.
- If Git helps to maintain a Linux kernel project of that magnitude operating smoothly, you can imagine how easy git can make collaboration on your projects.

Git Vs Github

- Git is a distributed version control system, it is a tool to manage your project source code history.

- Whereas Github is a web based, git file hosting service which enables us to showcase/share our projects and files to others.



Git Installation

```
root@IITB:/git# git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
```

```
root@IITB:/git# git --version
git version 2.14.1
root@IITB:/git#
```

Git Configuration

```
root@IITB:/git# git config --global user.name "Prof. B Thangaraju"
root@IITB:/git# git config --global user.email "b.thangaraju@iiitb.ac.in"
root@IITB:/git# git config --list
user.name=Prof. B Thangaraju
user.email=b.thangaraju@iiitb.ac.in
root@IITB:/git#
```

```
root@IITB:/git/project1# ls -a
.  ..
root@IITB:/git/project1# git init
Initialized empty Git repository in /git/project1/.git/
root@IITB:/git/project1# ls -al
total 12
drwxr-xr-x 3 root root 4096 Feb 19 18:33 .
drwxr-xr-x 7 root root 4096 Feb 19 18:31 ..
drwxr-xr-x 7 root root 4096 Feb 19 18:33 .git
root@IITB:/git/project1# cd .git
root@IITB:/git/project1/.git# ls -l
total 32
drwxr-xr-x 2 root root 4096 Feb 19 18:33 branches
-rw-r--r-- 1 root root   92 Feb 19 18:33 config
-rw-r--r-- 1 root root   73 Feb 19 18:33 description
-rw-r--r-- 1 root root   23 Feb 19 18:33 HEAD
drwxr-xr-x 2 root root 4096 Feb 19 18:33 hooks
drwxr-xr-x 2 root root 4096 Feb 19 18:33 info
drwxr-xr-x 4 root root 4096 Feb 19 18:33 objects
drwxr-xr-x 4 root root 4096 Feb 19 18:33 refs
root@IITB:/git/project1/.git#
```

Git demo

```
1 public class HelloWorld {
2     public static void main( String[] args ) {
3             System.out.println( "Hello World from Master Branch !" );
4             System.exit( 0 );
5         }
6 }
```

```
root@IITB:/git/branch_demo# git init
Initialized empty Git repository in /git/branch_demo/.git/
root@IITB:/git/branch_demo# git add HelloWorld.java
root@IITB:/git/branch_demo# git commit -m "First commit from Master"
[master (root-commit) 742e5b3] First commit from Master
 1 file changed, 6 insertions(+)
 create mode 100644 HelloWorld.java
root@IITB:/git/branch_demo# git branch
* master
root@IITB:/git/branch_demo# mylog
* 742e5b3 (HEAD -> master) First commit from Master
```

Java Demo Program

```java
import java.util.Scanner;
 public class caladd {
    public static void main(String args[]) {
       float a, b, res;
       char choice, ch;
       Scanner scan = new Scanner(System.in);

        do {
           System.out.print("1. ADD TWO Numbers\n");
           System.out.print("2. Exit\n\n");
           System.out.print("Enter Your Choice : ");
           choice = scan.next().charAt(0);
           switch(choice){
              case '1' : System.out.print("Enter Two Number : ");
                         a = scan.nextFloat();
                         b = scan.nextFloat();
                         res = a + b;
                         System.out.print("Result = " + res);
                         break;

              case '2' : System.exit(0);
                         break;

              default : System.out.print("INVALID CHOICE!!!");
                        break;
           }
             System.out.print("\n---------------------------------------\n");
        }while(choice != 2);
```

```
root@IITB:/upgrad/cal# javac caladd.java
root@IITB:/upgrad/cal# java caladd
1. ADD TWO Numbers
2. Exit

Enter Your Choice : 1
Enter Two Number : 20 30
Result = 50.0
---------------------------------------
1. ADD TWO Numbers
2. Exit

Enter Your Choice : 2
root@IITB:/upgrad/cal#
```

Git –Stage 1

When you first make changes to a file, the changes will exist only on your local computer in your working directory. Git has not yet track the changes or modification to those files. In other words, these files and its changes are not yet part of your development history or are not visible to anyone except for you.

```
root@IITB:/git/project1# ls
caladd.java
root@IITB:/git/project1# git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        caladd.java

nothing added to commit but untracked files present (use "git add" to track)
```

Git –Stage 2: add

- git should add any files or changes that you want to include in your development history, so git knows which files and changes that it should track.
- The files that we add to our development history at this stage goes to something called the “staging area.”
- Staging area is used to review the files and files changes that you have made. After doing your reviews, you can then decide which changes you want git to permanently track.

```
root@IITB:/git/project1# git add caladd.java
root@IITB:/git/project1# git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   caladd.java

root@IITB:/git/project1#
```

Git – Stage 3: commit

Once you're sure about the staged files, you can make a record of your changes and git will remember the changes. This is known as a commit. The commit will now become a permanent part of your development history.

```
root@IITB:/git/project1# git commit -m "Adding Multiplication module" caladd.java
[master (root-commit) 68f544f] Adding Multiplication module
 1 file changed, 35 insertions(+)
 create mode 100644 caladd.java
root@IITB:/git/project1# git status
On branch master
nothing to commit, working tree clean
root@IITB:/git/project1# git log
commit 68f544f2d60e3c909e8cc2a0dcec666fc111e06f (HEAD -> master)
Author: Prof. B Thangaraju <b.thangaraju@iiitb.ac.in>
Date:   Mon Feb 19 19:35:31 2018 +0530

    Adding Multiplication module
root@IITB:/git/project1#
```

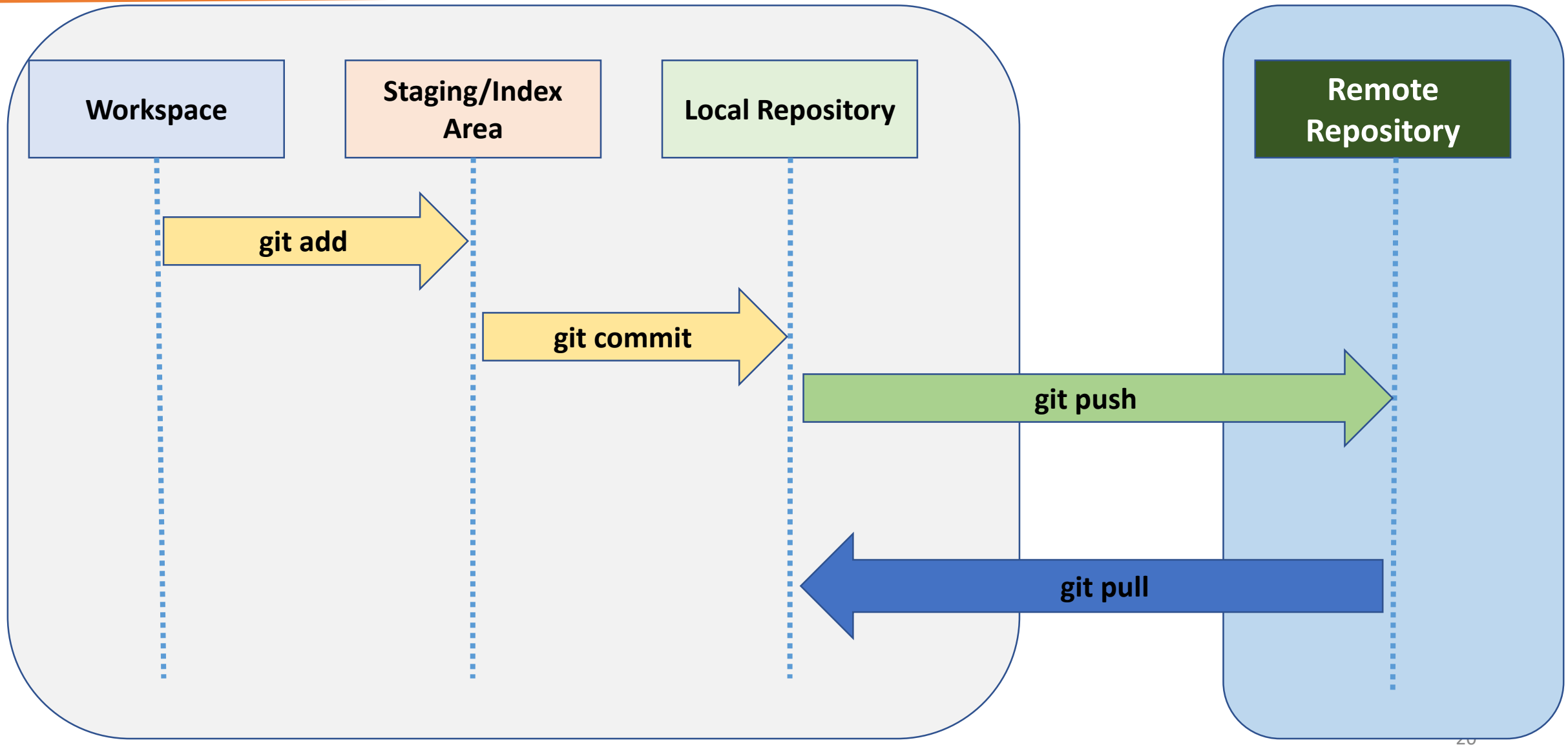
Git – Stage 3: commit

- There can be a number of commits for a project or code. Each commit might happen at a particular timestamp.
- A timeline of commits create a graph.
- Commits create links to other commits, forming a graph of your development history. You can revert back to your previous commit, see how files changed from one commit to the next, even fix bugs in any of the commits, and review information such as where and when changes were made.
- Commits are identified in Git by a unique id.
- Each time a commit is made in Git, that commit is assigned an unique id. Because everything has unique id , it is impossible to make changes, lose information, or corrupt files without Git detecting it.

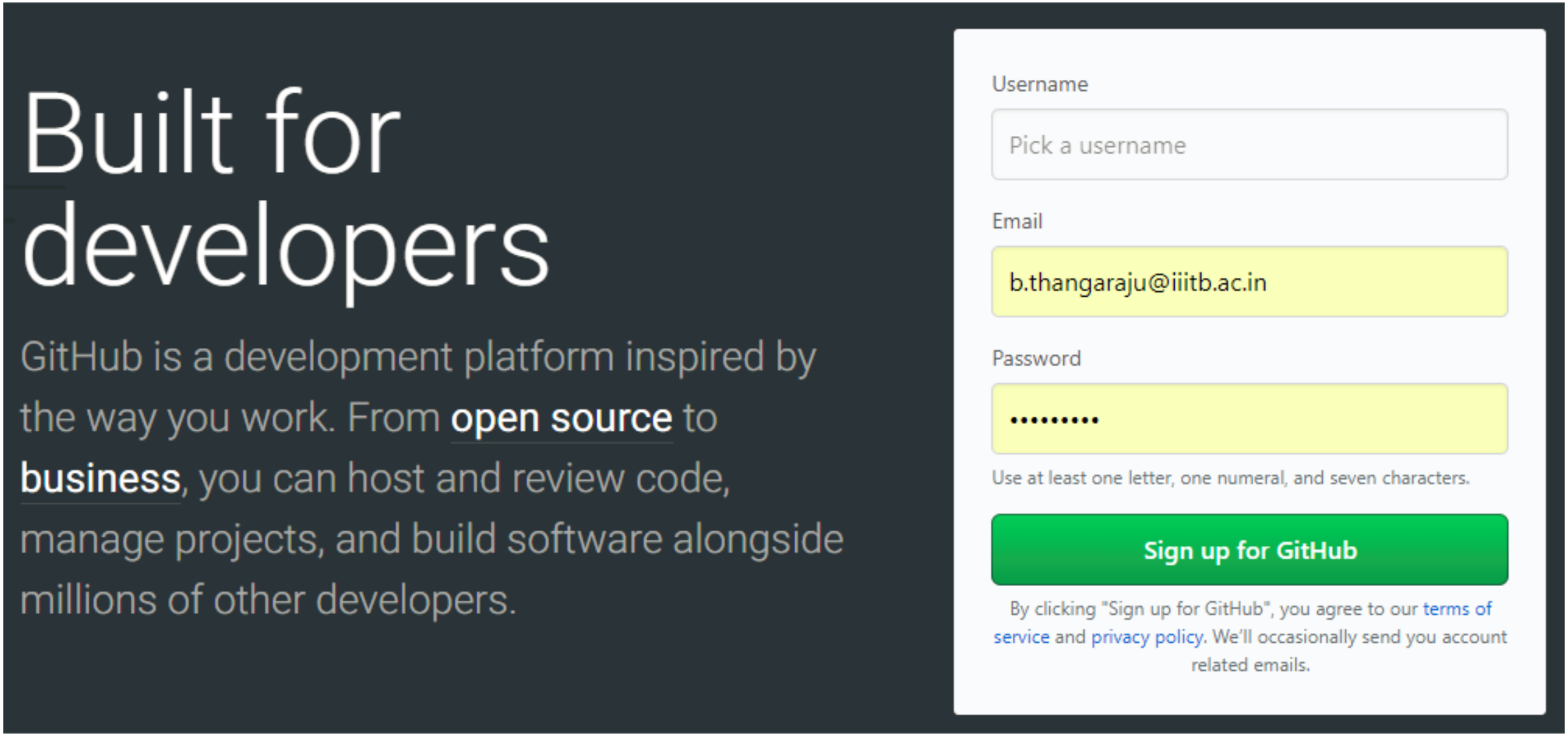
Git log

- We will run the command **#git log**
- This command will show us the entire commit history for the project
- It will show us the commit id, the username and mail id of person who made the changes and also the date and time when the commit was made
- Now we can see the commit ids, if we want to go back to a previous version or commit we can use that commit id and run the following command-
- **Git checkout <commit id>**

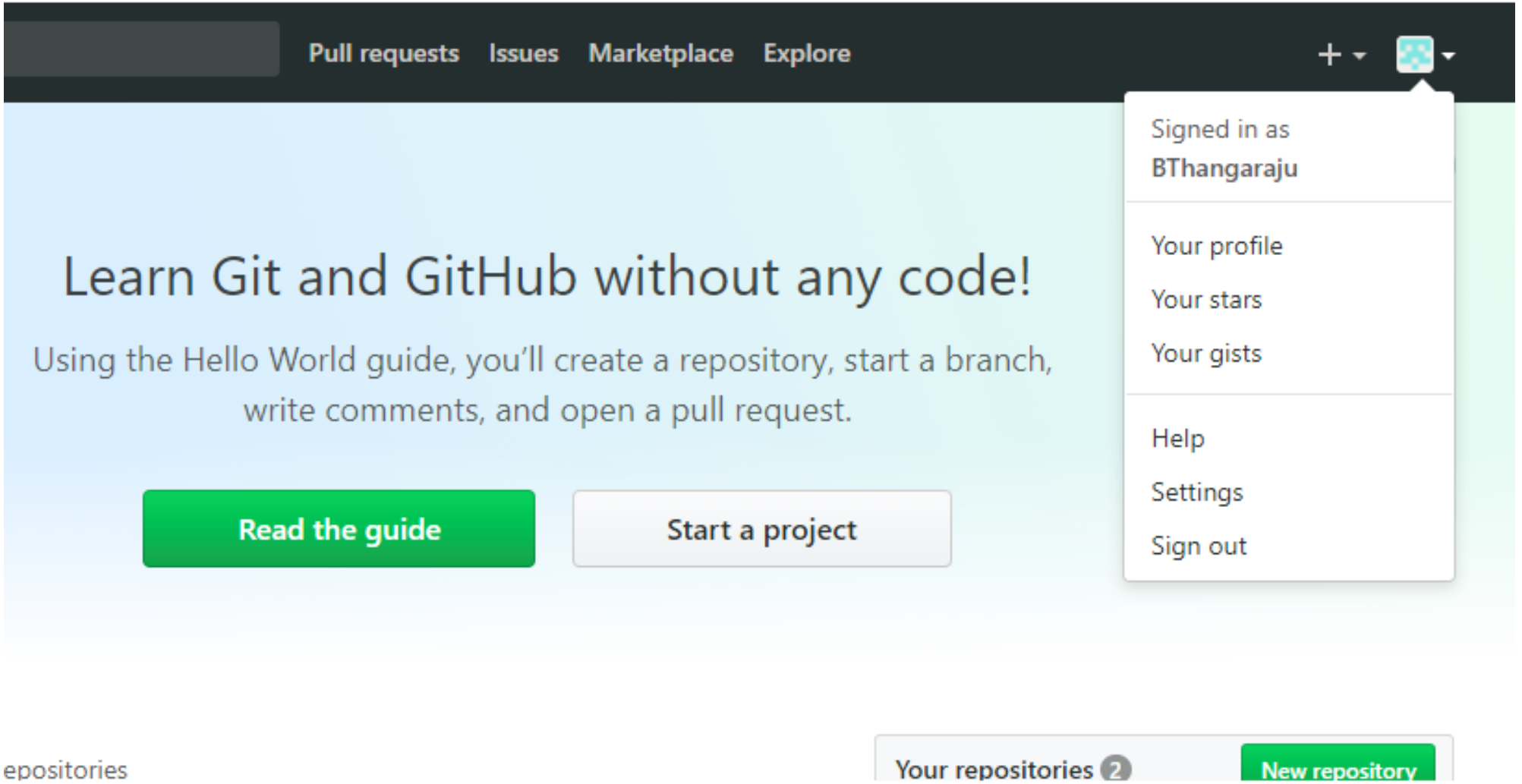
Git Workflow



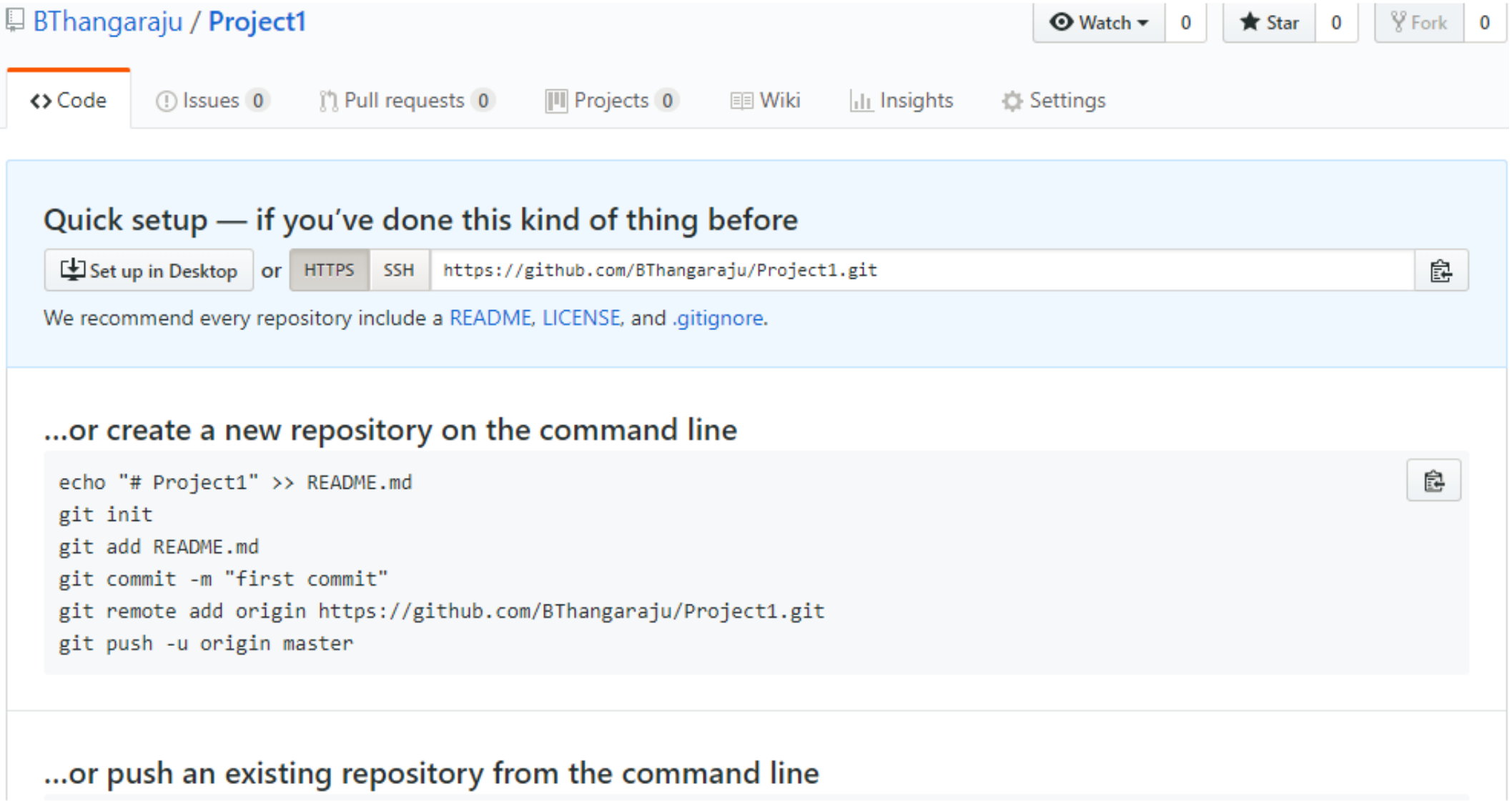
https://github.com/



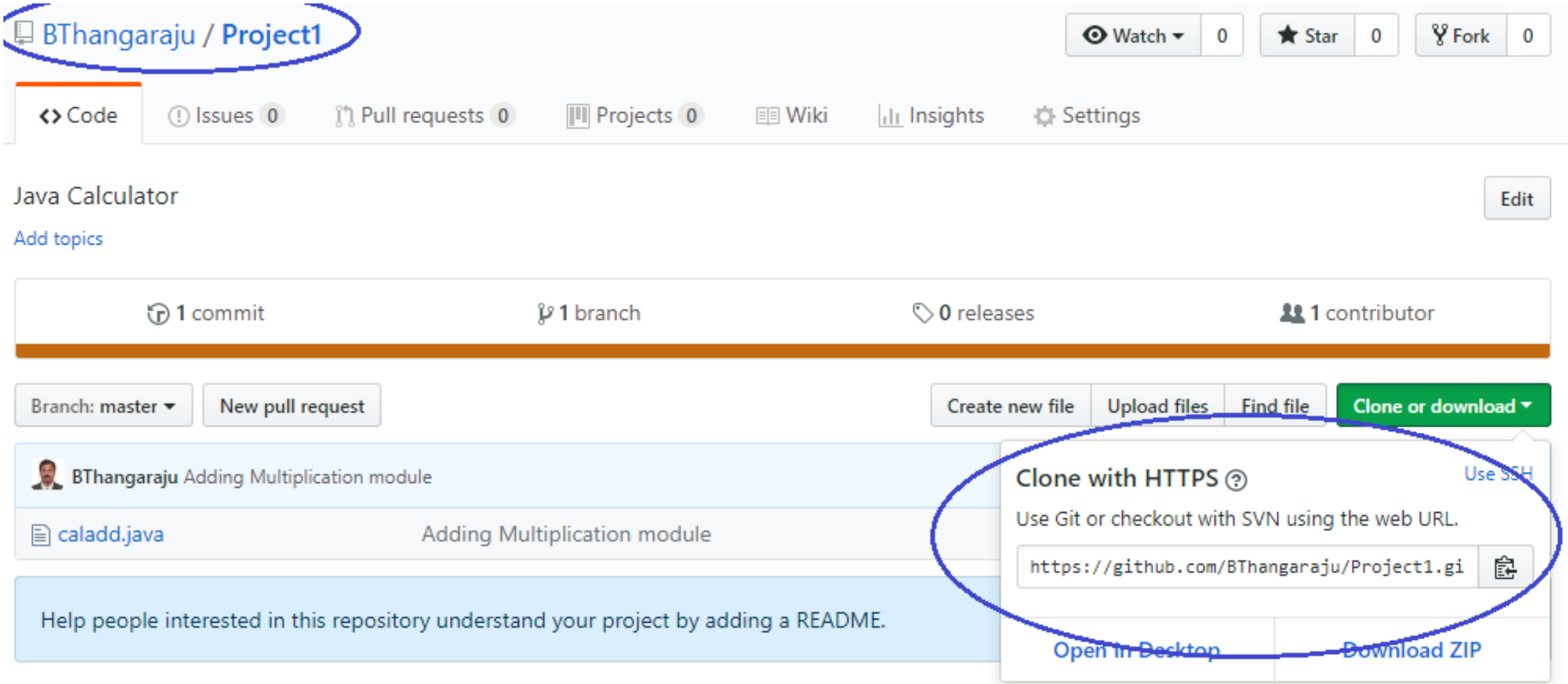
Sign up


Pull requests
Issues
Marketplace
Explore
Signed in as
BThangaraju
Your profile
Your stars
Your gists
Help
Settings
Sign out
Learn Git and GitHub without any code!
Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.
Read the guide
Start a project
epositories
Your repositories 2
New repository

Create a Project as Project1



Github url


BThangaraju / Project1
Watch 0
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Java Calculator
Edit
Add topics
1 commit
1 branch
0 releases
1 contributor
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
BThangaraju Adding Multiplication module
caladd.java
Adding Multiplication module
Help people interested in this repository understand your project by adding a README.
Clone with HTTPS
Use SSH
Use Git or checkout with SVN using the web URL.
https://github.com/BThangaraju/Project1.gi
Open in Desktop
Download ZIP

Git remote and git push

```
root@IITB:/git/project1# git remote add origin https://github.com/BThangaraju/Project1.git
root@IITB:/git/project1# git push -u origin master
Username for 'https://github.com': BThangaraju
Password for 'https://BThangaraju@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 645 bytes | 322.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/BThangaraju/Project1.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
root@IITB:/git/project1#
```

git remote add origin <URL>

- This command gathers all the committed files from your local repository and uploads them to our repository we created on github.

git push -u origin master

- This command will push all our local changes to our online repository that is our repository on Github.

“Git push” tells git that we want to upload our files and development history from the repository on our local computer to a repository hosted on github

“-u origin master” means that we want to upload the “master” version of our commits to a repository on github called “origin”

Git branches

- A branch is the fundamental means of launching a separate line of development with a software project. In git, you can create many branches resulting in many different lines of development within a git repository. The branch management in git is lightweight and simple to learn.
- Each team in a project can work on a different branch simultaneously. When you are ready and if you want to merge some specific branches or all the branches with master branch you can do it easily.
- To start off with, git will automatically create a master copy (or “branch” in git terms) of your project when you create a repository. This master copy is called the “master” branch.
- If you want to do parallel development with the existing project code, without making any changes to our master branch then you can create different branches based on your need and each branch will have the same copy of the master branch project source code.
  - For example, you can create different branches for different team members. Or you can create different branches for the different features that you are adding to the project.

Working with Git branches

1. Create branches
2. View the created branches
3. Work concurrently with different branches
4. Merge the different branches with the master branch
5. Delete the created branches.

To create a branch we can use command **#git branch branchname**

#git branch -this command will show us all the created branches

You can work concurrently with different branches

- If we want to move from one branch to other we can run the command, **#git checkout branchname**

Working with Git branches

if we are on our master branch and we want to move to branch1, we will use the command #git checkout branch1

- This command will change our branch from master to branch 1
- Next we will add comment into the calculator.java code.

```
root@IITB:/git/branch_demo# more HelloWorld.java
/* This is Hello World Java proram - team1 added the comment */
public class HelloWorld {
    public static void main( String[] args ) {
            System.out.println( "Hello World from Master Branch !" );
            System.exit( 0 );
        }
}
root@IITB:/git/branch_demo# git add HelloWorld.java
root@IITB:/git/branch_demo# git commit -m "Added comment by team1"
[team1 f816c7f] Added comment by team1
 1 file changed, 1 insertion(+)
root@IITB:/git/branch_demo#
```

since we are working in branch 1, calculator.java will be added into the “branch1” branch and cannot be seen from the “master” branch

- Next we will checkout to Branch2
- Next we will again Add comment into the calculator.java code
- And now we will add and commit the file into the branch2.

```
root@IITB:/git/branch_demo# git checkout team2
Switched to branch 'team2'
root@IITB:/git/branch_demo# vim HelloWorld.java
root@IITB:/git/branch_demo# mylog
* f816c7f (team1) Added comment by team1
* 742e5b3 (HEAD -> team2, master) First commit from Master
root@IITB:/git/branch_demo# git add HelloWorld.java
root@IITB:/git/branch_demo# git commit -m "Added print statement by team2"
[team2 ce4f3ae] Added print statement by team2
 1 file changed, 1 insertion(+)
root@IITB:/git/branch_demo#
```

- After this we will move to our master branch and check the log message, which shows a detailed information about our commit details in an order.

```
root@IITB:/git/branch_demo# git checkout master
Switched to branch 'master'
root@IITB:/git/branch_demo# git log --all
commit ce4f3ae9e8d5e63c064178be4113a0a5c83f5152 (team2)
Author: Prof. B Thangaraju <b.thangaraju@iiitb.ac.in>
Date:   Tue Feb 20 18:39:22 2018 +0530

    Added print statement by team2

commit f816c7f7d421c35f9083d709cf666aaad9dbba27 (team1)
Author: Prof. B Thangaraju <b.thangaraju@iiitb.ac.in>
Date:   Tue Feb 20 18:36:09 2018 +0530

    Added comment by team1

commit 742e5b349db4bd96513544fff42106094c14f890 (HEAD -> master)
Author: Prof. B Thangaraju <b.thangaraju@iiitb.ac.in>
Date:   Tue Feb 20 18:04:41 2018 +0530

    First commit from Master
root@IITB:/git/branch_demo#
```

Merging branches

- **Merging different branches into master branch**
- Now, we can use the command **#git merge branchname to** merge branch1 into the master and then merge team2 branch into the master branch.
- ***#git merge branch1***
- ***#git merge branch 2***
- Now if we want to the see the difference between the lines of code from branc1 and master we will use the command
- ***#git diff master..branch1***
- *This* command will highlight the added lines into the calculator.java file.

```
root@IITB:/git/branch_demo# git merge team1
Updating 742e5b3..f816c7f
Fast-forward
 HelloWorld.java | 1 +
 1 file changed, 1 insertion(+)
root@IITB:/git/branch_demo# git merge team2
Auto-merging HelloWorld.java
Merge made by the 'recursive' strategy.
 HelloWorld.java | 1 +
 1 file changed, 1 insertion(+)
root@IITB:/git/branch_demo# javac HelloWorld.java
root@IITB:/git/branch_demo# java HelloWorld
Hello World from Master Branch !
Hello World from team2 branch !
root@IITB:/git/branch_demo# more HelloWorld.java
/* This is Hello World Java proram - team1 added the comment */
public class HelloWorld {
    public static void main( String[] args ) {
            System.out.println( "Hello World from Master Branch !" );
            System.out.println( "Hello World from team2 branch !" );
            System.exit( 0 );
        }
}
root@IITB:/git/branch_demo#
```

Deleting Branches

Deleting branches locally

- To delete the created branches, we use command, **#git branch -d branch_name**

```
root@IITB:/git/branch_demo# git branch
* master
  team1
  team2
root@IITB:/git/branch_demo# git branch -d team1
Deleted branch team1 (was dee8ea9).
root@IITB:/git/branch_demo# git branch -d team2
Deleted branch team2 (was db228db).
root@IITB:/git/branch_demo# git branch
* master
root@IITB:/git/branch_demo#
```

Managing Conflict

When a team is working on the same project on the same files but from different branches conflicts might arise maybe two people end up changing the same lines of code in a given file.

Git will be confused about which one of the conflicting changes it should consider for merging. And in such cases when there are merge conflict, the merge will fail, and we won't be able to merge one branch into another branch.

If branch1 and branch2 modified the same line in the calculator.java program then it will create conflict when you want to merge the branches into the master branch.

```
root@IITB:/git/branch_demo# git diff master..team1
diff --git a/HelloWorld.java b/HelloWorld.java
index bd89cbc..5b4f403 100644
--- a/HelloWorld.java
+++ b/HelloWorld.java
@@ -1,6 +1,7 @@
 public class HelloWorld {
     public static void main( String[] args ) {
             System.out.println( "Hello World!" );
+            System.out.println( "Hello World! program modified by team1" );
             System.exit( 0 ); //success
         }
 }
root@IITB:/git/branch_demo# git diff master..team2
diff --git a/HelloWorld.java b/HelloWorld.java
index bd89cbc..9511e84 100644
--- a/HelloWorld.java
+++ b/HelloWorld.java
@@ -1,6 +1,7 @@
 public class HelloWorld {
     public static void main( String[] args ) {
             System.out.println( "Hello World!" );
+            System.out.println( "Hello World! program modified by team2" );
             System.exit( 0 ); //success
         }
 }
root@IITB:/git/branch_demo#
```

```
root@IITB:/git/branch_demo# git checkout master
Switched to branch 'master'
root@IITB:/git/branch_demo# git merge team1
Updating 99e18cb..dee8ea9
Fast-forward
 HelloWorld.java | 1 +
 1 file changed, 1 insertion(+)
root@IITB:/git/branch_demo# git merge team2
Auto-merging HelloWorld.java
CONFLICT (content): Merge conflict in HelloWorld.java
Automatic merge failed; fix conflicts and then commit the result.
root@IITB:/git/branch_demo#
```

Managing Conflict

- The first merge between master and branch1 is working fine but the second merge with team2 creates conflict.
- When you open the calculator.java file, we can see the conflict details and you can decide whether you want to keep both the changes or remove any modifications created by branch2.

```
public class HelloWorld {
    public static void main( String[] args ) {
        System.out.println( "Hello World!" );
<<<<<< HEAD
        System.out.println( "Hello World! program modified by team1" );
=======
        System.out.println( "Hello World! program modified by team2" );
>>>>>>> team2
        System.exit( 0 ); //success
    }
}
```

Managing Conflict

- To remove a conflict we should use our best judgments to resolve branch conflicts.

We should use our best judgment to determine which branch is “correct” and which branch is “faulty”

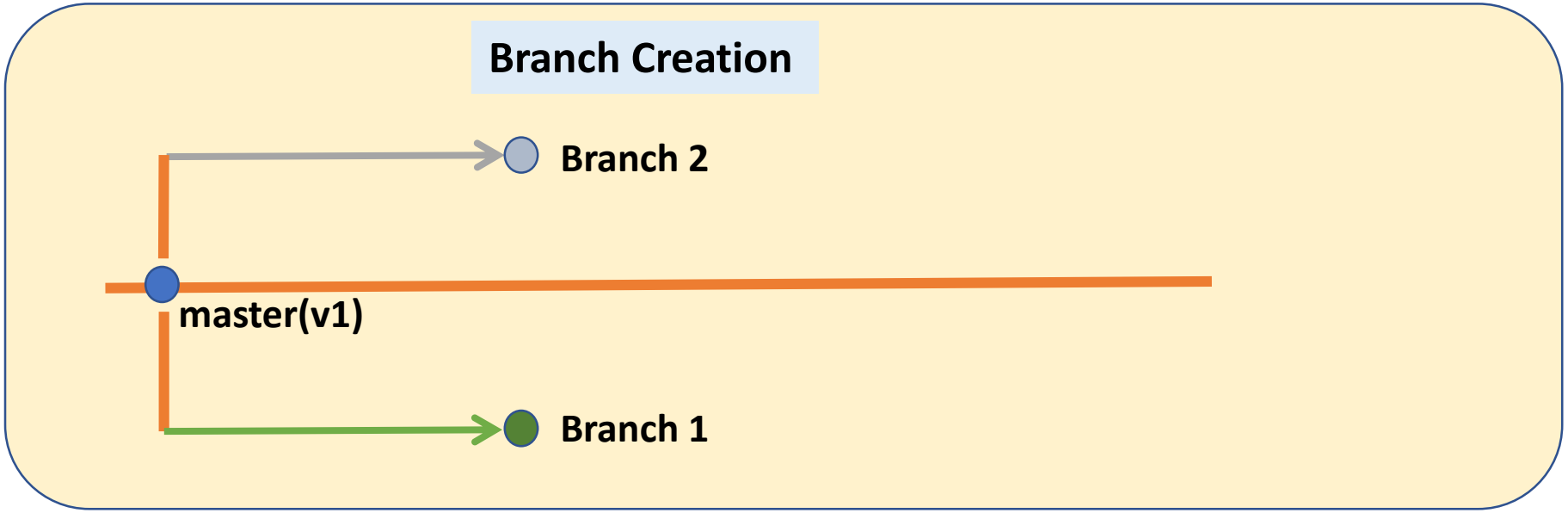
- Now removed the HEAD and team2 lines and save the file and then commit the file to keep modifications done by team1 and team2.

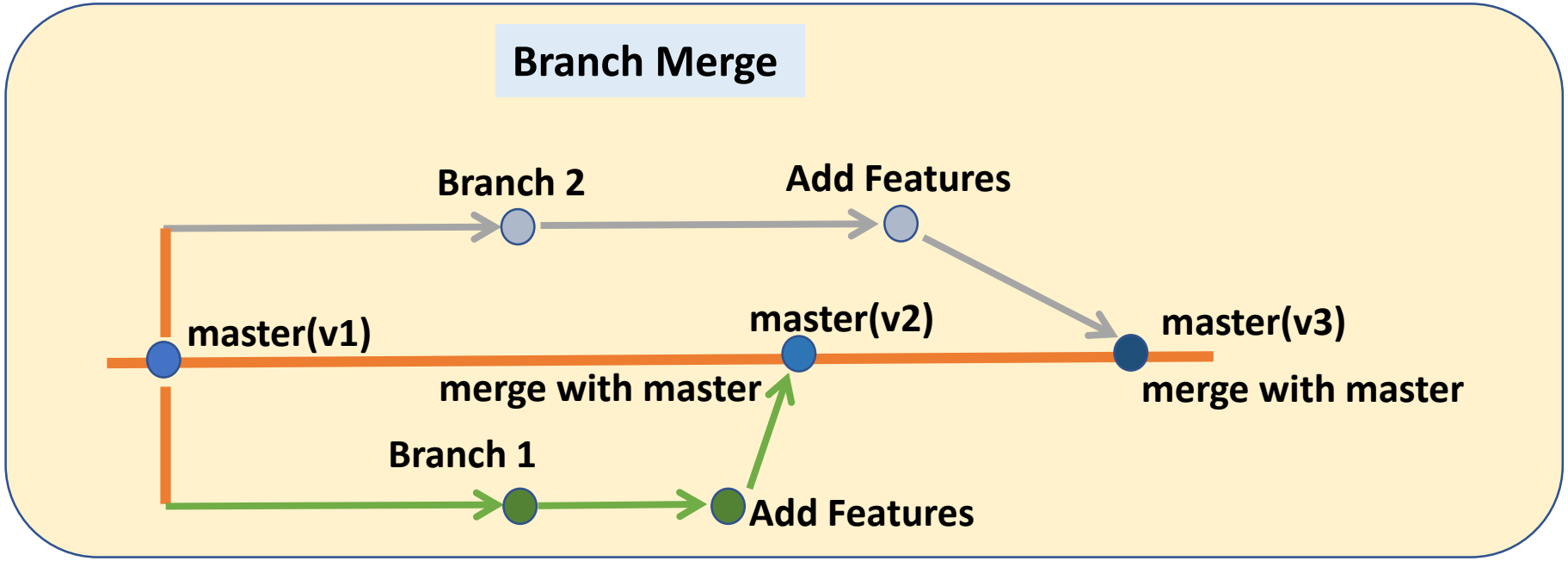
Then add and commit the calculator.java file into the master branch.

When you compile and execute the program, it will be working fine and can see the modifications done by both branch1 and branch 2

```
root@IITB:/git/branch_demo# vim HelloWorld.java
root@IITB:/git/branch_demo# git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   HelloWorld.java

no changes added to commit (use "git add" and/or "git commit -a")
root@IITB:/git/branch_demo# git add HelloWorld.java
root@IITB:/git/branch_demo# git commit -m "Resolved the conflict"
[master ae6036c] Resolved the conflict
root@IITB:/git/branch_demo# javac HelloWorld.java
root@IITB:/git/branch_demo# java HelloWorld
Hello World!
Hello World! program modified by team1
Hello World! program modified by team2
root@IITB:/git/branch_demo#
```

git branch creation and merging





Git clone –working with remote repo

A clone is a copy of a repository. A clone contains all the objects from the original. Each clone is an independent and autonomous repository and a symmetric peer of the original.

- **git clone <URL> -** The command git clone creates a new, local Git repository on your computer and copies all the files, commit histories, commit messages, branches, and etc.

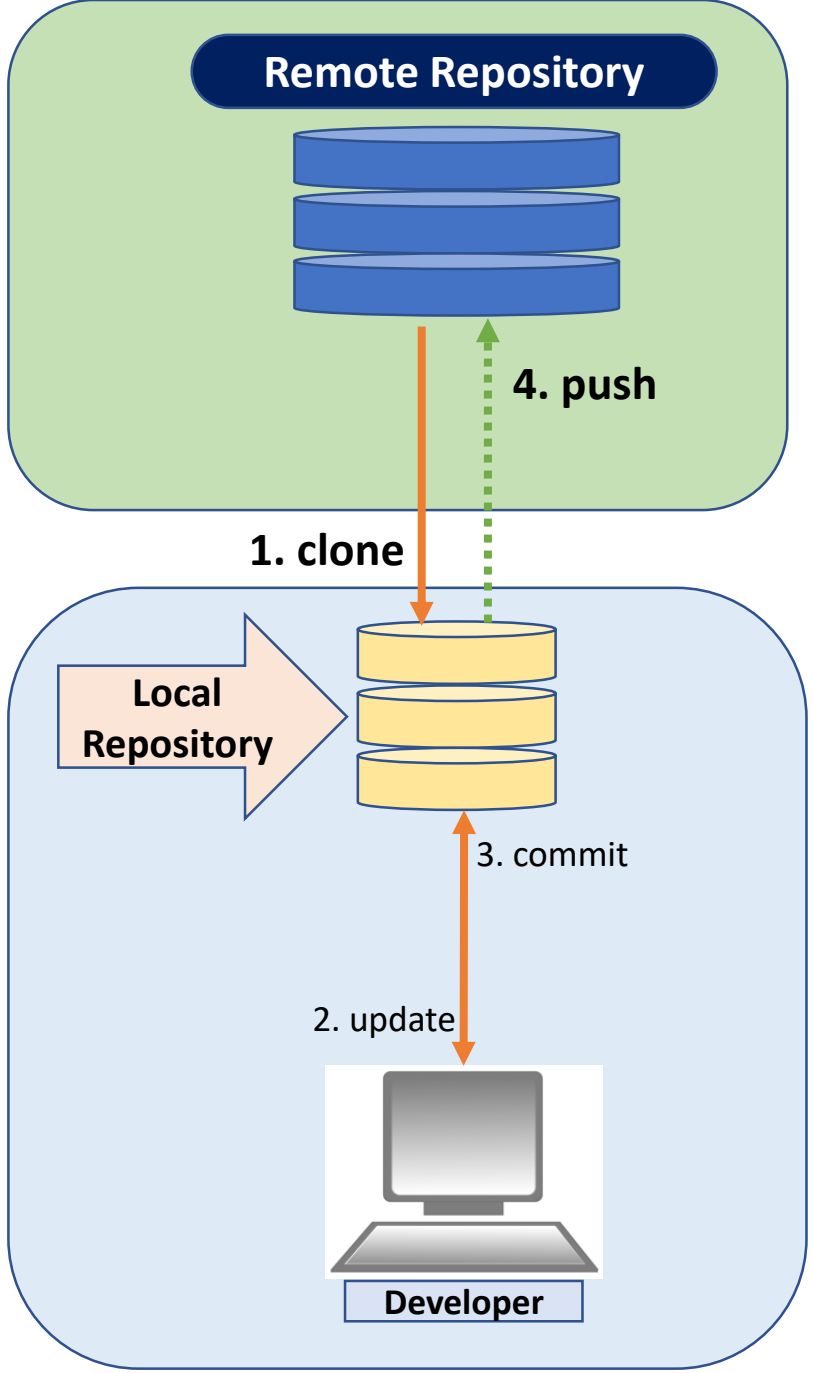
```
root@IITB:/git/local# git clone https://github.com/BThangaraju/Project1.git
Cloning into 'Project1'...
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
root@IITB:/git/local# ls
Project1
root@IITB:/git/local# cd Project1/
root@IITB:/git/local/Project1# ls
caladd.java
root@IITB:/git/local/Project1#
```

Git clone –working with remote repo

```
root@IITB:/git/local/Project1# git add caladd.java
root@IITB:/git/local/Project1# git commit -m "Added many features in the project"
[master 836ef19] Added many features in the project
 1 file changed, 34 insertions(+), 11 deletions(-)
root@IITB:/git/local/Project1# git push https://github.com/BThangaraju/Project1
Username for 'https://github.com': BThangaraju
Password for 'https://BThangaraju@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 527 bytes | 527.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/BThangaraju/Project1
   68f544f..836ef19  master -> master
root@IITB:/git/local/Project1#
```

- To verify the new commit in github, refresh your **github project1** page and you can see the new updated calladd.java file.

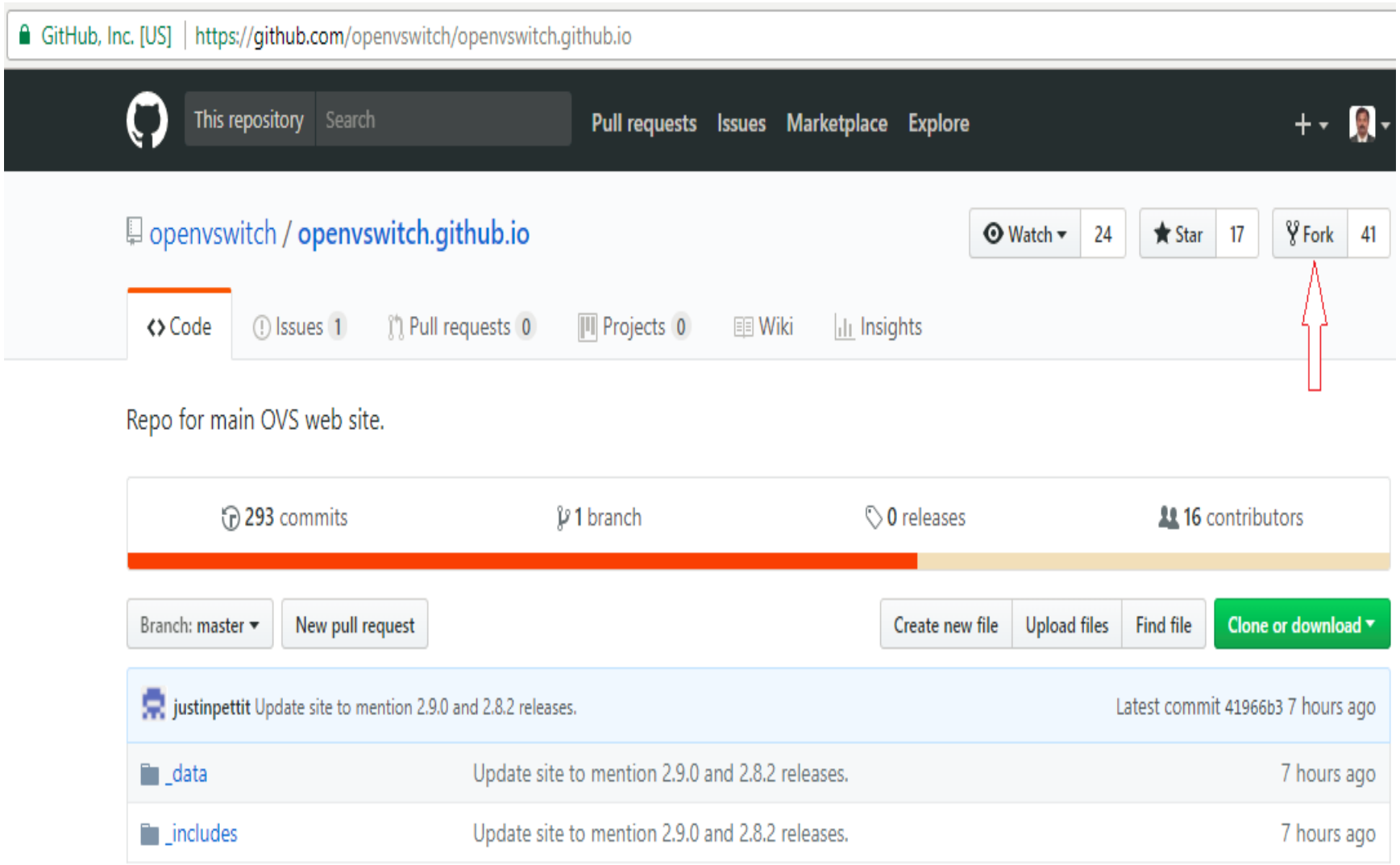
git clone


Remote Repository
4. push
1. clone
Local Repository
3. commit
2. update
Developer

Git fork

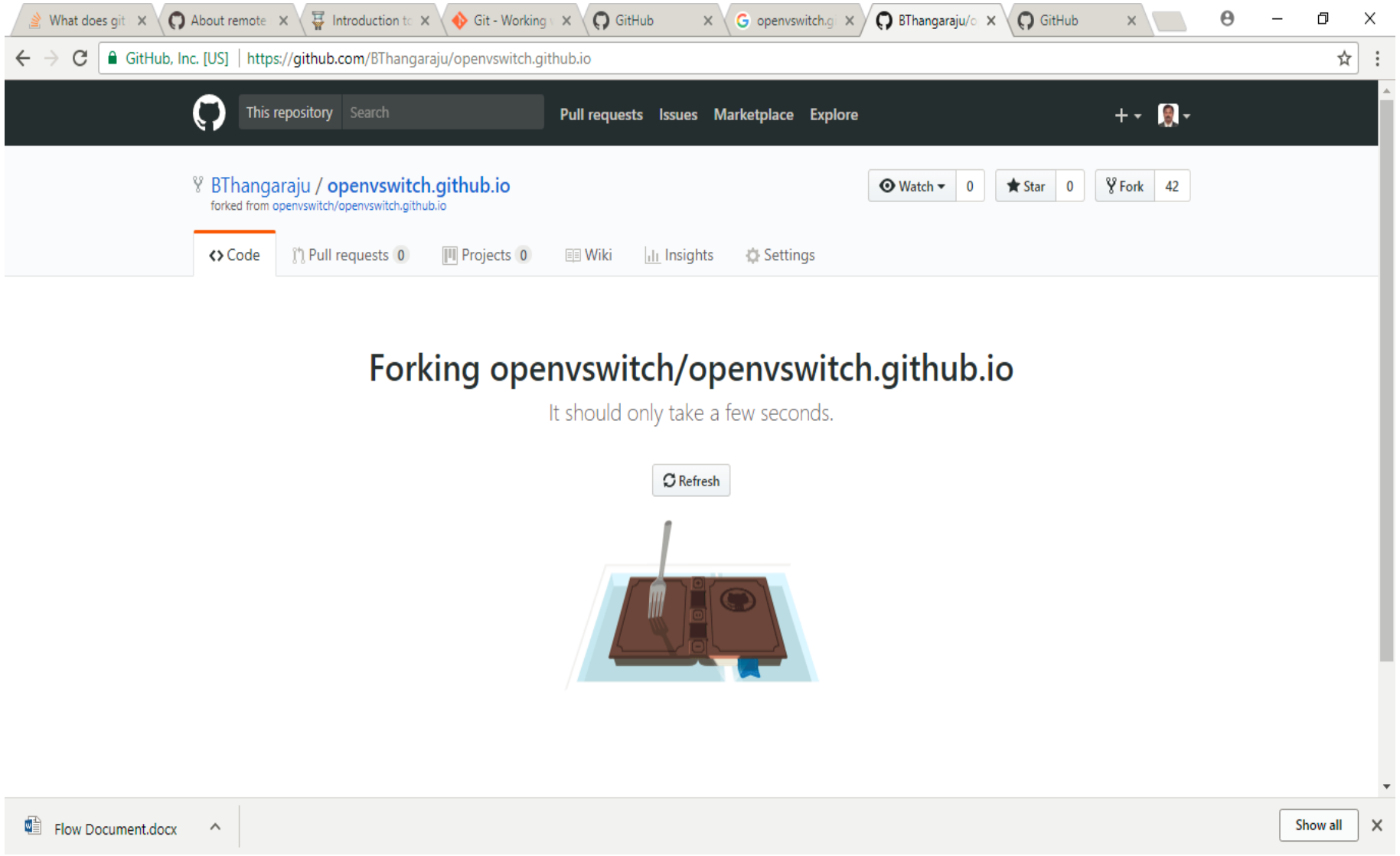
- If you want to work with someone projects or if I want to collaborate with any open source projects then I need to go their github web page and fork their project into my github account.
- For example, in our demo, I will fork OVS project from https://github.com/openvswitch/openvswitch.github.io
- OVS (Open v Switch) is an open source implementation of a distributed virtual multilayer switch.
- OVS is to provide a switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in computer networks.
- In the following page, click fork button

Collaborate with OSS Project



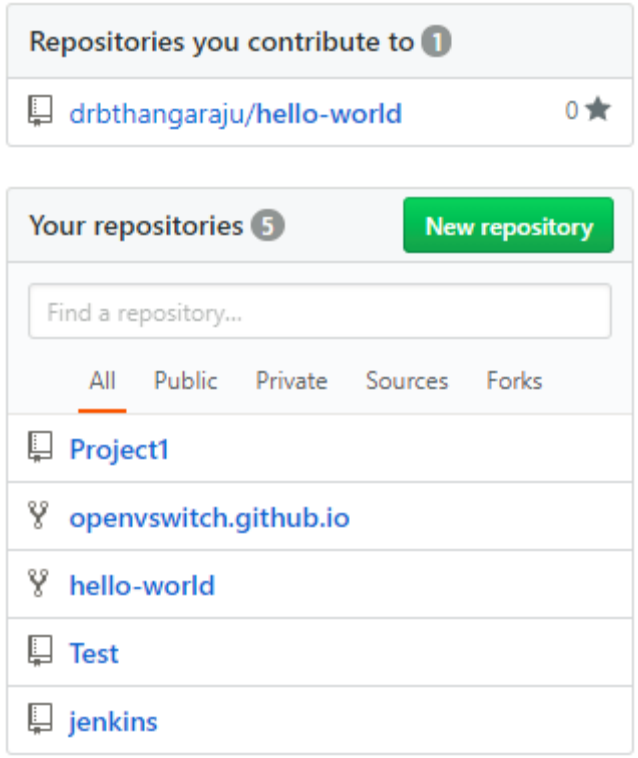
Collaborate with OSS Project

1.



Collaborate with OSS Project

Now I will go to my github account and check the forked repository.

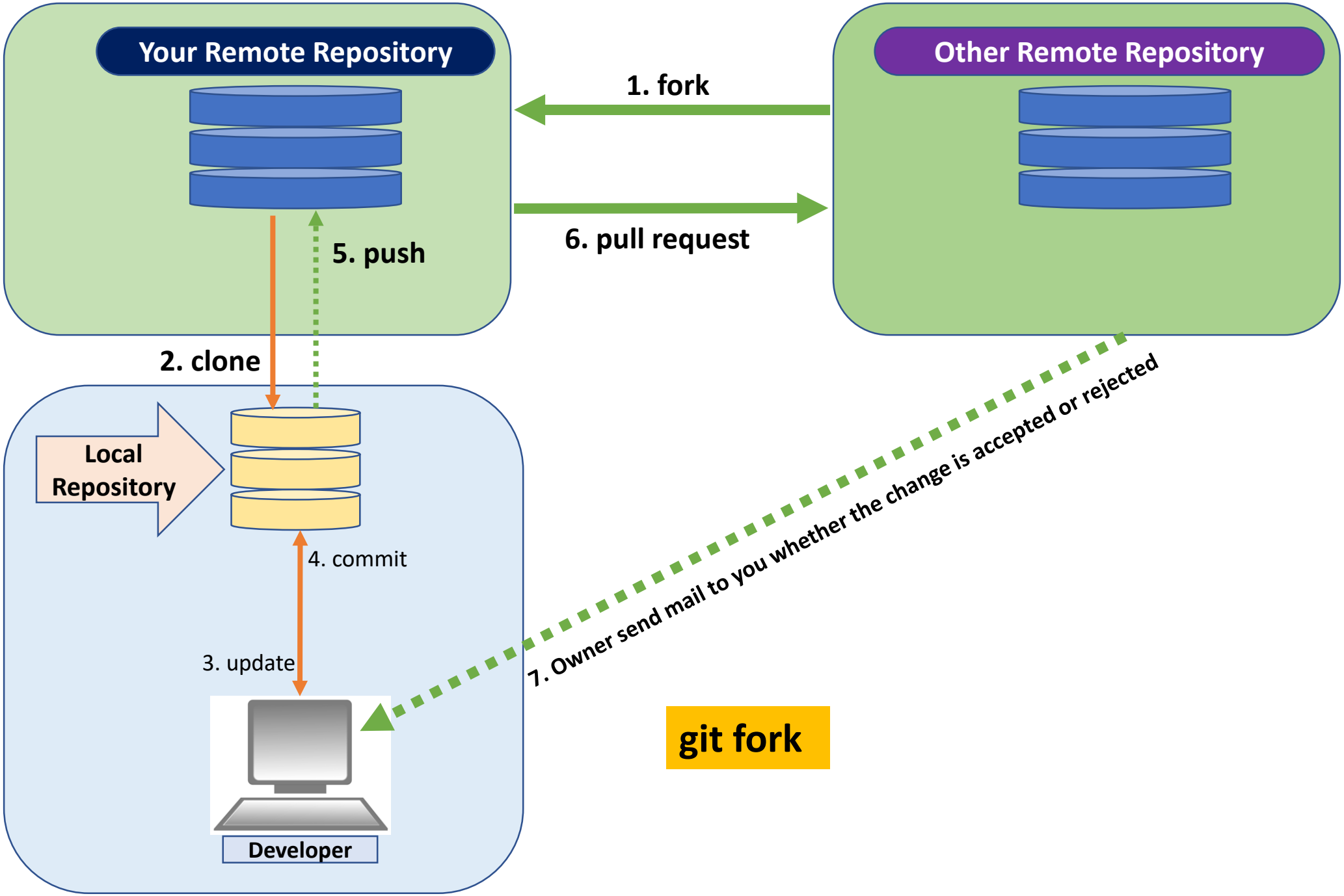


The **openvswitch** open source project is in my remote repository.

You can do the modifications or add any new changes into the project after cloning into the local repository and push the changes into your remote repository.

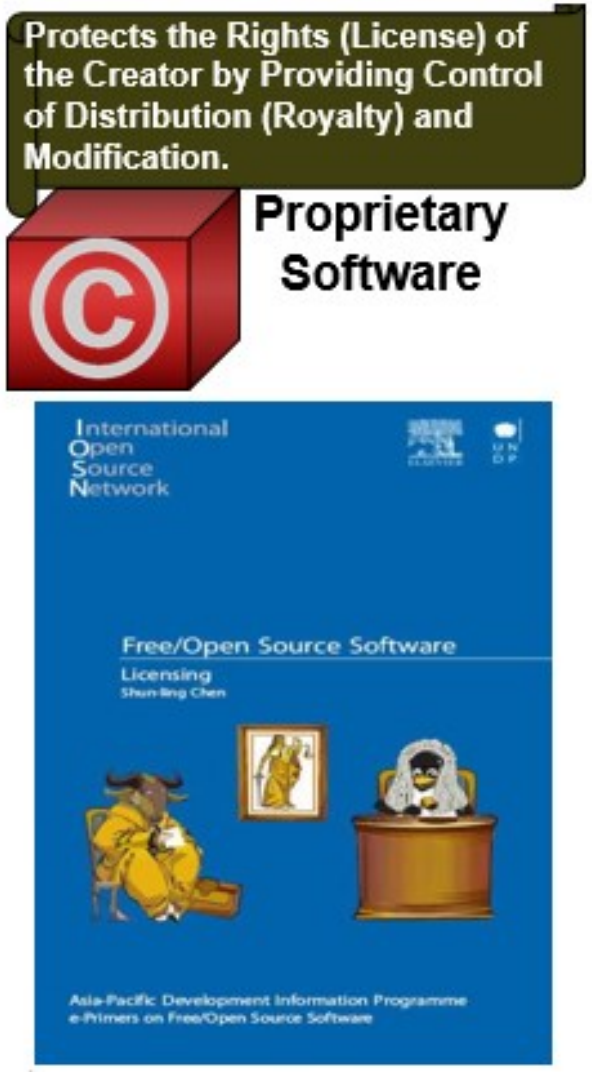
If you want to contribute back to the openvswitch project, you can create **pull request** with propose file change details.

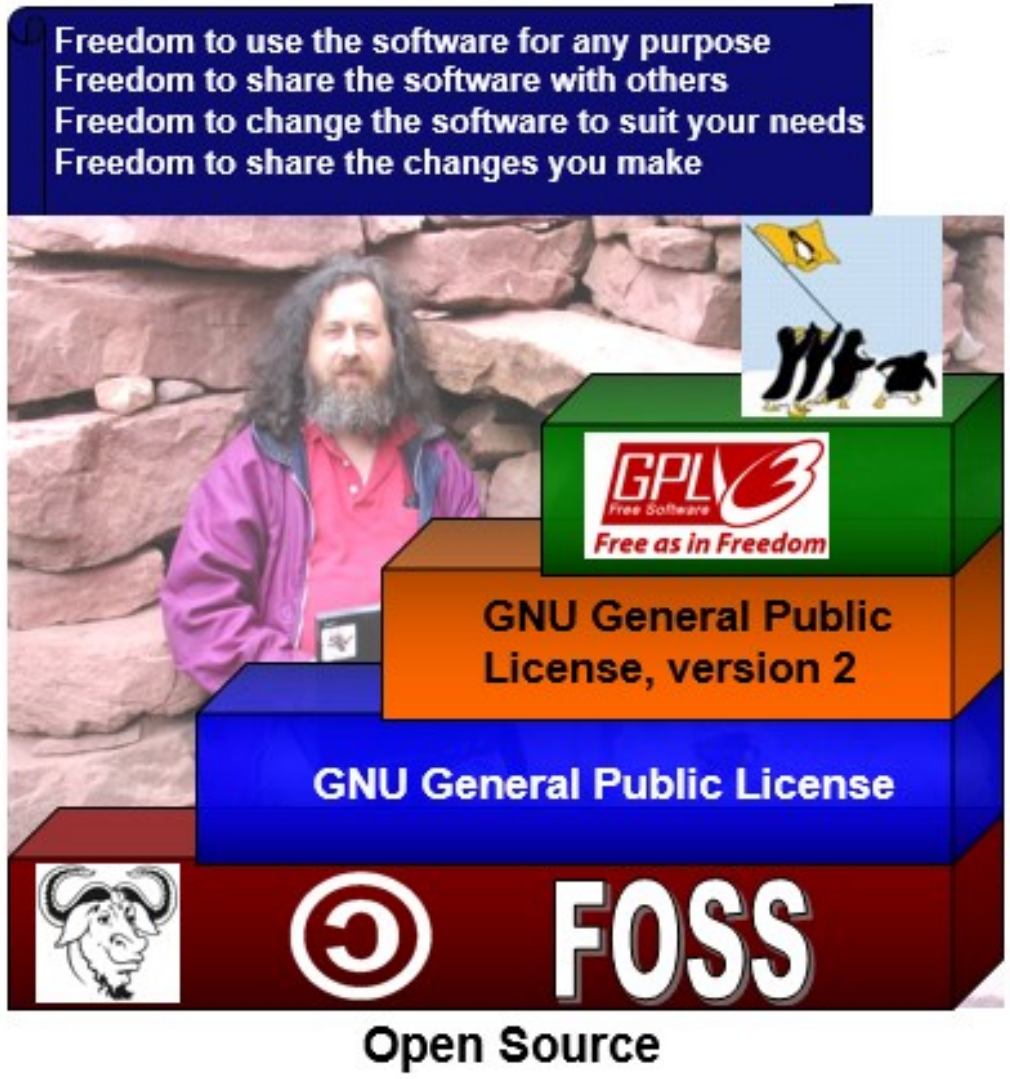
The project maintainer of the openvswitch will decide whether will accept or reject your change proposal.


Your Remote Repository
Other Remote Repository
1. fork
6. pull request
5. push
2. clone
Local Repository
4. commit
3. update
Developer
7. Owner send mail to you whether the change is accepted or rejected
git fork

Working with OSS

- ***open source*** refers to any program whose ***source*** code is made available for use or modification as users or other developers see fit.
- ***Open source*** software is usually developed as a public collaboration and made freely available.
- The logo for open source is copy left symbol and the license is GPL (General Public License). One of the main founders of open source is **Richard Stallman.**



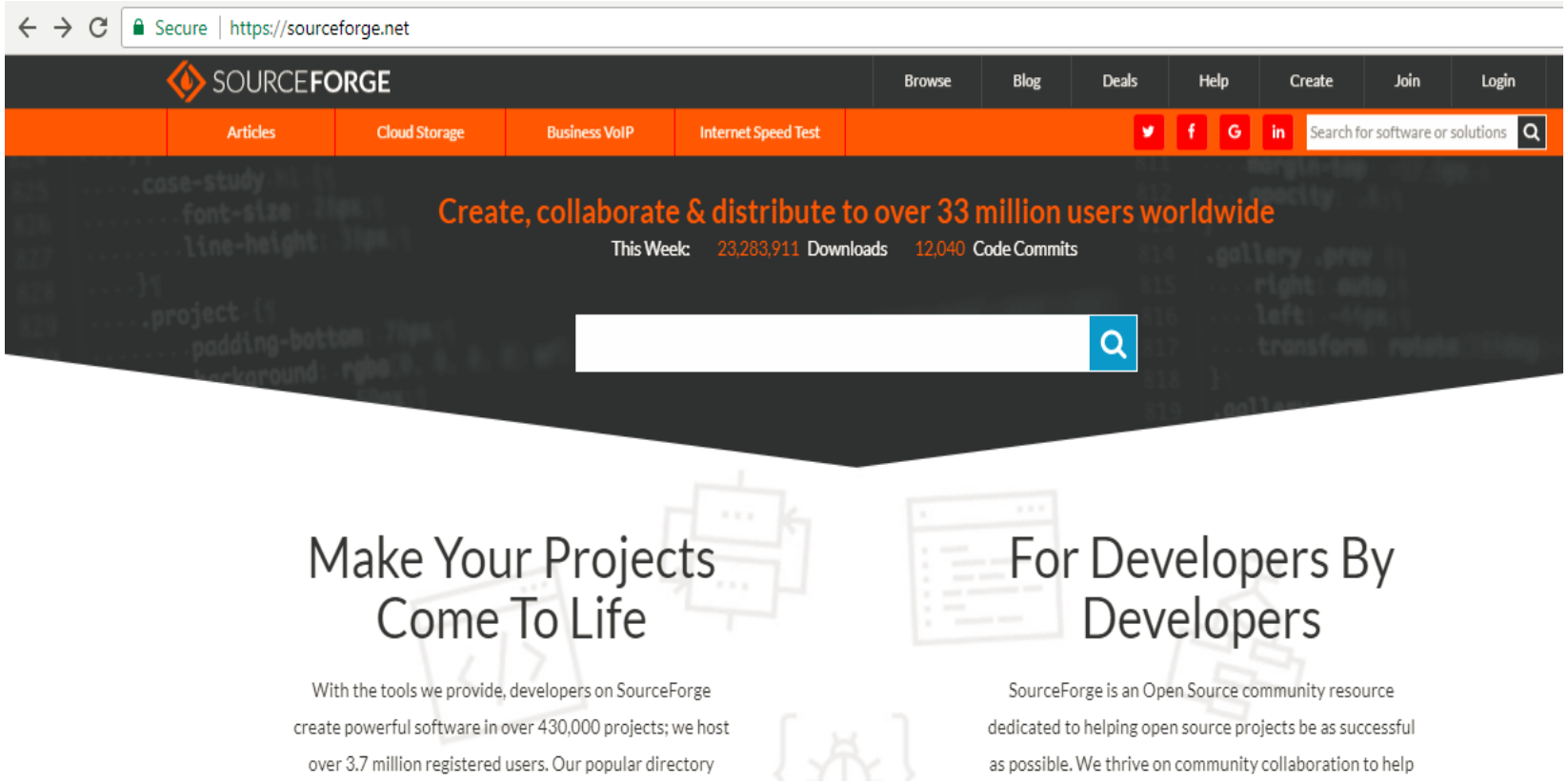


Working with OSS

Linux kernel (core part of the Linux OS) is published in open source at kernel.org, many developers develop their own version of Linux operating system in sourceforge.net. Below is the some list of Linux distribution.

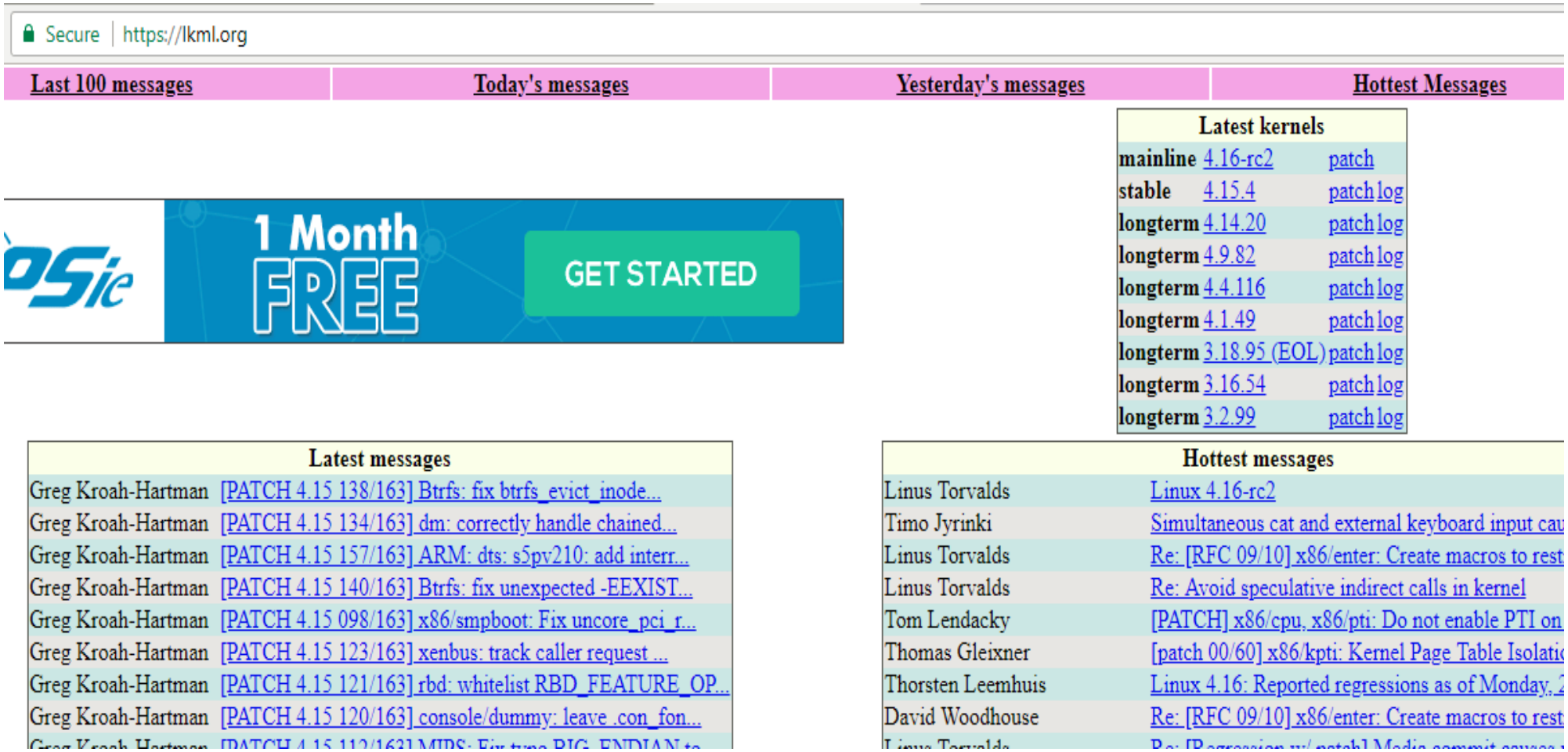
Working with OSS

You can browse the sourceforge.net website and you can view open source projects in different categories. If you are interested you can also work with any projects or you can create your own project and collaborate with others.



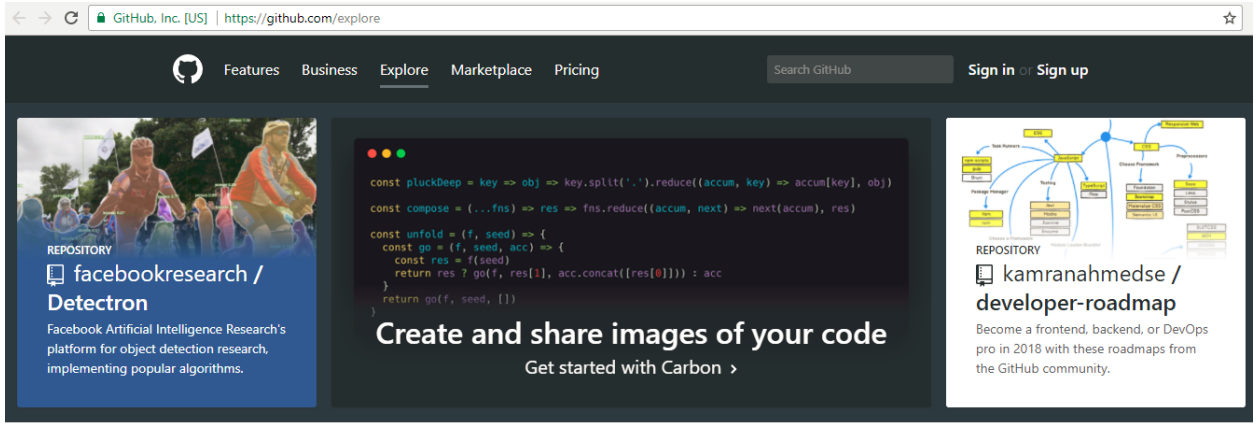
Working with OSS

If you are interested to work with Linux kernel, you can view the Kernel mailing list home page.



Working with OSS

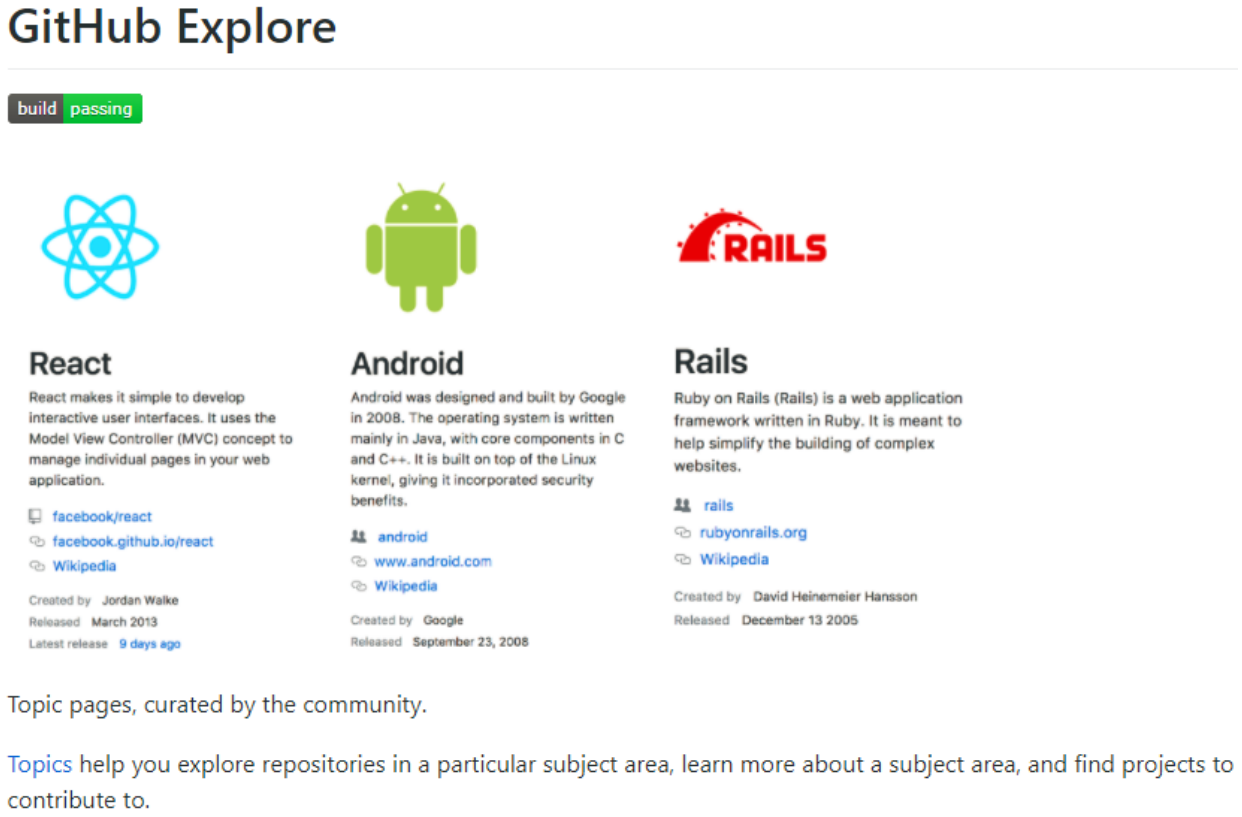
- You can browse the most used topics in Github, you can find at: https://github.com/topics.
- You can contribute to GitHub's set of featured topics in the github/explore repository. It categories different topics of your interest.



Starring a repository shows appreciation to the repository maintainer for their work.

GitHub's repository rankings depend on the number of stars a repository has.

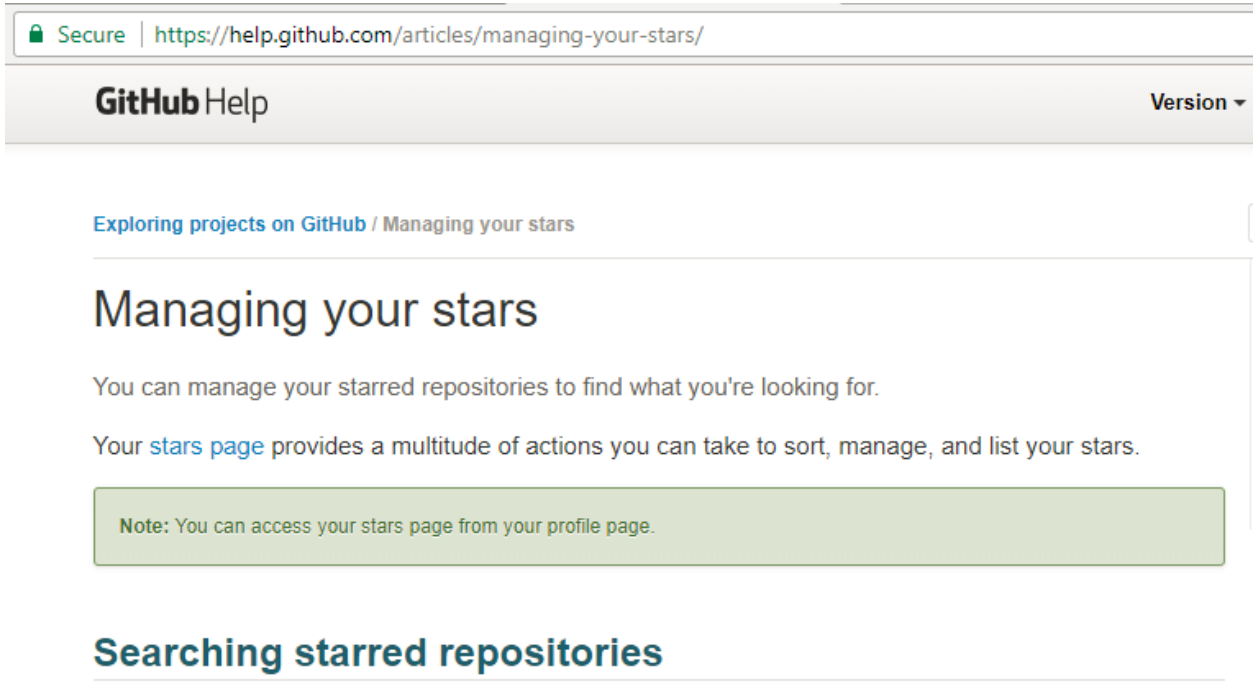
You can view all the repositories you have starred in your stars page.



Working with OSS

The stars manage page gives details about how to managing your stars.

- github as a community where your repositories are public and it allows other coding enthusiasts to view your projects and if they are interested they will also help you to add new features into your project or fix if there are any issues.
- So anyone who is interested in hiring you for a job or maybe an internship, you can give them a link to your github portfolio.
- They can view all the projects you have worked on. If you are really active on github it will give potential employers a chance to see your work and you will get a better chance to get suitable job opportunities.



THANK YOU