Git Notes v1.0

PART-B

SHAILESH YADAV
B.E EXTC
Shaileshyadav7958@gmail.com

Table of Contents

Chap1: What is Branch & need of it?

We will discuss below point's:

What is branching?
What is the need of creating a new branch?
Various commands related to branching?
Demo example
Multiple scenarios where branching is required?
Advantages of branches?
Merge operation
How to resolve merge conflict?

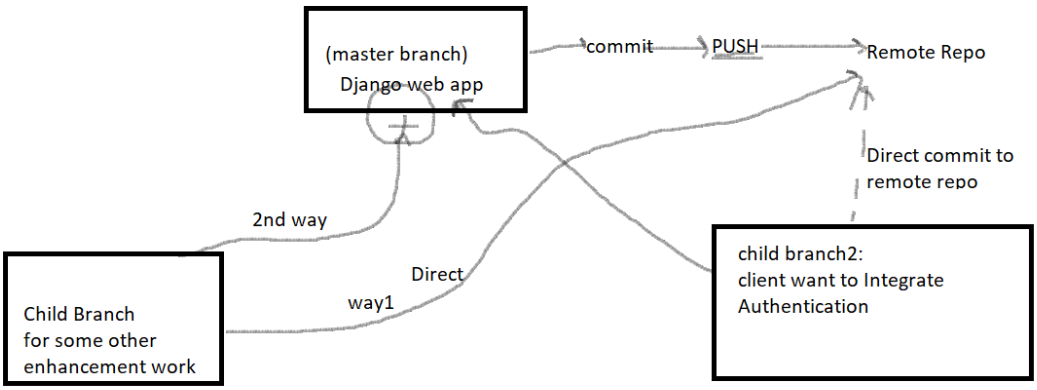
It is most Important concept 😊
Mandatory concept and unavoidable concept if we are working on real time project.

Till now whatever we did in PART-A learning master branch is default branch/main branch in git.

```
$ git branch
* master
```

Generally main source code we will place in master branch.

So now question, what is the need of creating a branch? 😊



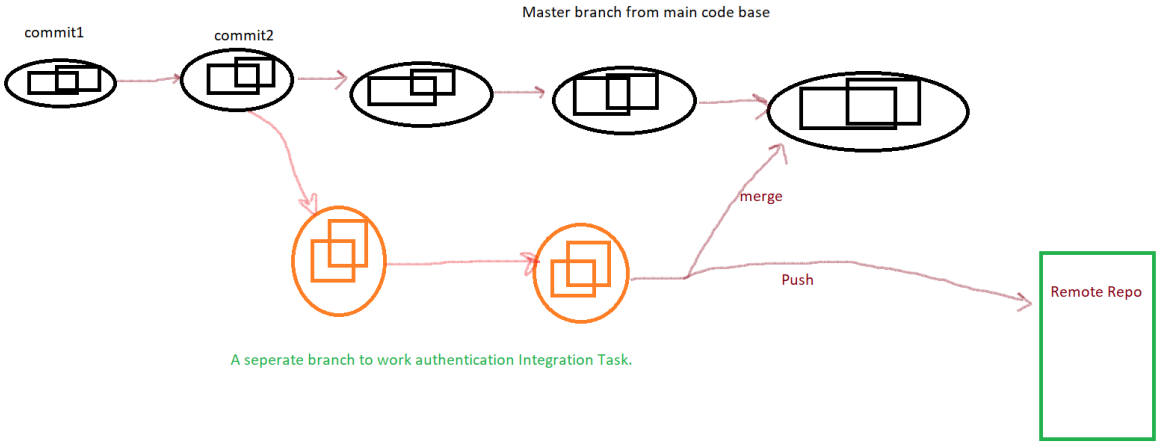
Note: Suppose client came and he want some enhancement work no problem sir we will create new branch (May take 15-90 days)right so once done we can directly PUSH it to remote repo OR merge with master and then commit and push

branch and all are
Independent of each
otehrs.

Que: Why not Separate workspace instead of separate branch?
Ans: It is because If we are creating a separate Workspace for enhancement project then after completion we need to merge it manually but in case of branch automatic merging support are there which'll take few seconds and also It will help in less memory use and time saving 😊.

//Benefit: Multiple flows, Parallel development, Code base will be very clean.
Conclusion:

1. Once we create a child branch then all commits, and all files will be inherited from parent branch.
2. In child branch we can create new file and make changes in existing file and can commit those changes based on our requirement.
3. All branches are isolated (completely independent)- Changes in one branch will not be visible to other branch. (After creating child branch if we create any files in master branch that will not be available to child 😊 because It's isolated.)
4. Once Task completed in Child branch, we'll have two option first is directly merge it with master and commit and second is we can PUSH that branch directly to the remote repository.
5. From below dig we can see that we have master branch and a separate branch which we created for some enhancement work. Once it is done, we can either merge it with master branch OR can push it to remote repo.
6. An Independent flow of work is nothing but branching 😊.



Chap2-a: Various Git command related with branching

Various commands related to branching:

a) To view available branches:

```
  $ git branch
* master
```

//here * means current working branch.

Note: Even we can use git status command also to know working branch.

```
$ git status
On branch master
nothing to commit, working tree clean
```

---

b) To create a new branch:

No problem, sir same command with branch name 😊.

```
$ git branch Authentication_Work
```

//To cross check

```
$ git branch
  Authentication_Work
* master
```

---

c) How to switch from one branch to another branch?

We have already to git checkout (to discard un-staged (which is not in staging area) change in working directory).

Syntax: git checkout branch-name

```
$ git checkout Authentication_Work
Switched to branch 'Authentication_Work'

//To Verify

$ git branch
* Authentication_Work
  master


//Other way
$ git status
On branch Authentication_Work
nothing to commit, working tree clean
```

Note: Sir we are using git branch-name ➔to create and git checkout branch-name ➔to switch.
What if we to do both things in single go?

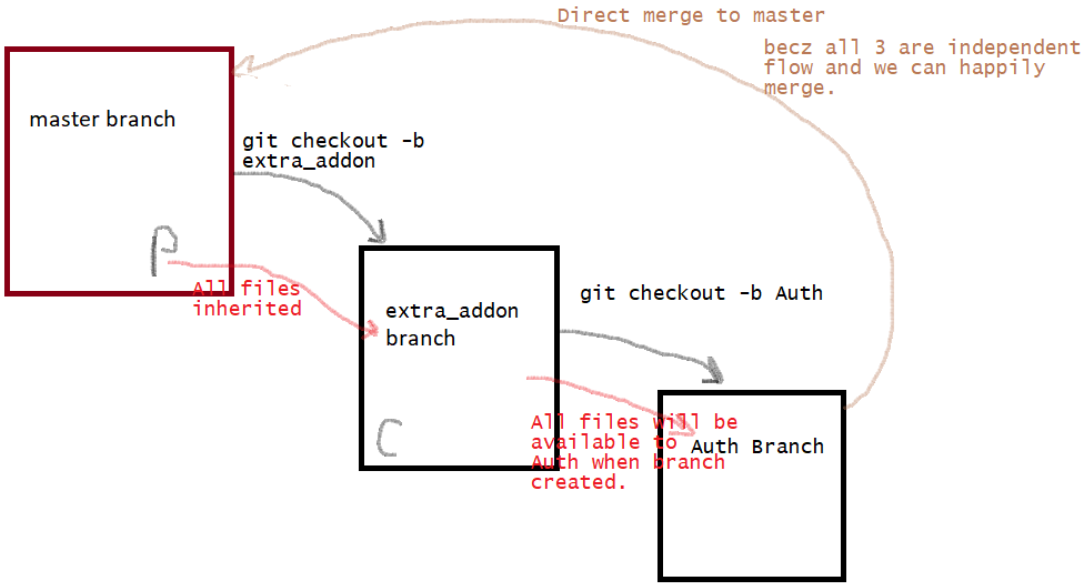
git branch-name ➔to create
git checkout branch-name ➔to switch

Shortcut Way:
Git checkout -b branch-name
//It will create branch-name and switch to that branch in single command.

```
$ git checkout -b extra_addon
Switched to a new branch 'extra_addon'
```

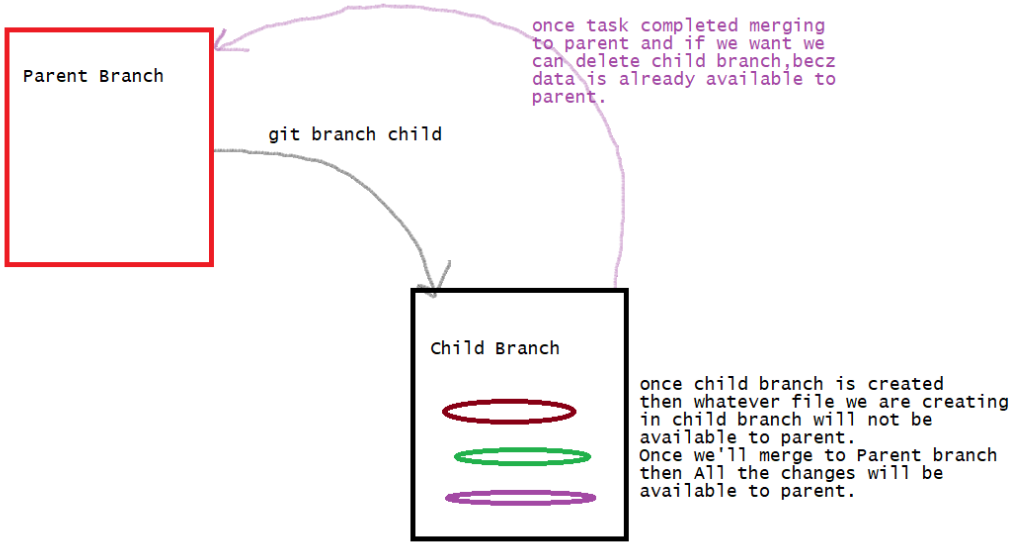
Let's verify

```
$ git branch
  Authentication_Work
* extra_addon
  master
```



Chap2-b: Demo on branching

Note: As we know whenever we are creating any branch then all the files and commit of that branch will be automatically available to the newly created branch. 😊.




```
$ mkdir branching_project
$ cd branching_project/
$ git init
Initialized empty Git repository in D:/Git Project/branching_project/.git/
$ touch a.txt b.txt c.txt
```

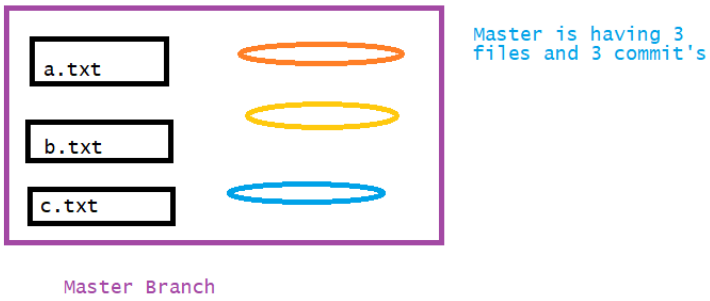
//let us add all above files to staging area and then commit it.()let me create 3 separate commit 😊.

```
$ git add a.txt; git commit -m "a.txt added."
[master (root-commit) 9965ce1] a.txt added.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt

$ git add b.txt; git commit -m "b.txt added."
[master 148ce7e] b.txt added.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 b.txt

$ git add c.txt; git commit -m "c.txt added."
[master c270e52] c.txt added.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c.txt

$ ls;git ls-files
a.txt  b.txt  c.txt
a.txt
b.txt
c.txt

$ git log --oneline
c270e52 (HEAD -> master) c.txt added.
148ce7e b.txt added.
9965ce1 a.txt added.
```

How many branches are there currently?

Only one
```
$ git branch
* master
```

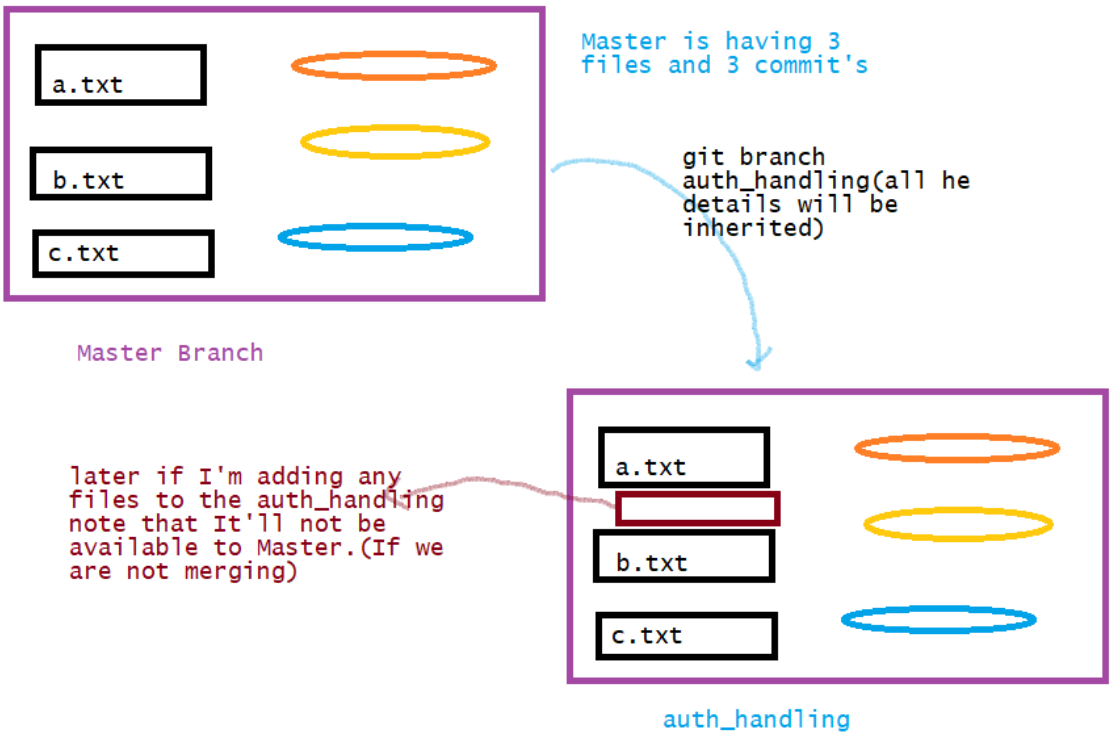
So, status in master



Now customer requirement is to add some new features so instead of implementing it inside master we'll create separate branch and work there.

```
$ git branch auth_handling
```
//created a new branch auth_handling

//let us switch to new branch from master (so master branch files and all commits will be automatically available to the auth_handling branch).




```
$ git checkout auth_handling
Switched to branch 'auth_handling'
```

```
$ git branch
* auth_handling
  master
```

So let check content of it:

```
/D/Git Project/branching_project (auth_handling)
$ ls;git ls-files
a.txt  b.txt  c.txt
a.txt
b.txt
c.txt
```

```
$ git log --oneline
c270e52 (HEAD -> auth_handling, master) c.txt added.
148ce7e b.txt added.
9965ce1 a.txt added.
```

//Now we can create a new file in child (auth_handling also).

```
/D/Git Project/branching_project (auth_handling)
$ touch x.txt y.txt z.txt
```

Now suppose we have completed our work so let me commit it.

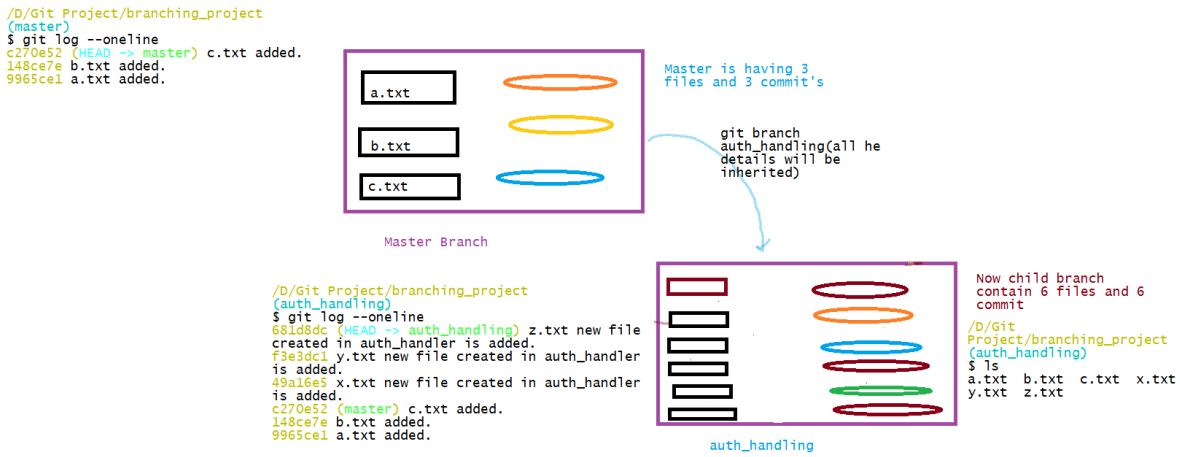
```
$ git status
On branch auth_handling
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        x.txt
        y.txt
        z.txt
nothing added to commit but untracked files present (use "git add" to track)
```

```
$ git ls-files
a.txt
b.txt
c.txt
$ ls
a.txt  b.txt  c.txt  x.txt  y.txt  z.txt
```

```
/D/Git Project/branching_project (auth_handling)
$ git add x.txt; git commit -m "x.txt new file created in auth_handler is added."
[auth_handling 49a16e5] x.txt new file created in auth_handler is added.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 x.txt

/D/Git Project/branching_project (auth_handling)
$ git add y.txt; git commit -m "y.txt new file created in auth_handler is added."
[auth_handling f3e3dc1] y.txt new file created in auth_handler is added.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 y.txt

/D/Git Project/branching_project (auth_handling)
$ git add z.txt; git commit -m "z.txt new file created in auth_handler is added."
[auth_handling 681d8dc] z.txt new file created in auth_handler is added.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 z.txt
```

Now our child branch contains total 6 files and 6 commit's (3 new files & commit)

```
/D/Git Project/branching_project (auth_handling)
$ git log --oneline
681d8dc (HEAD -> auth_handling) z.txt new file created in auth_handler is added.
f3e3dc1 y.txt new file created in auth_handler is added.
49a16e5 x.txt new file created in auth_handler is added.
c270e52 (master) c.txt added.
148ce7e b.txt added.
9965ce1 a.txt added.
```



Note: file x, y, z.txt will not be shown in master. It will have only a, b, c.txt files. (Same we can verify by switching into master branch).

```
$ git checkout master
Switched to branch 'master'

/D/Git Project/branching_project (master)
$ ls
a.txt  b.txt  c.txt
```

// Now let me add d.txt and commit in master 😊.

```
$ touch d.txt

/D/Git Project/branching_project (master)
$ git add d.txt;git commit -m "a new d.txt file is added in master branch post
auth_handler branch creation "
[master 2ca0c71] a new d.txt file is added in master branch post auth_handler
branch creation
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 d.txt
```

Now in master we are having 4 files

```
$ ls
a.txt  b.txt  c.txt  d.txt
```

And no of commit is:

```
$ git log --oneline
2ca0c71 (HEAD -> master) a new d.txt file is added in master branch post
auth_handler branch creation
c270e52 c.txt added.
148ce7e b.txt added.
9965ce1 a.txt added.
```

Note: Now we have created d.txt and commit after auth_handling so it will not be available in auth_handling.

Proof:

```
/D/Git Project/branching_project (master)
$ git checkout auth_handling
Switched to branch 'auth_handling'

/D/Git Project/branching_project (auth_handling)
$ ls
a.txt  b.txt  c.txt  x.txt  y.txt  z.txt
```

So, there is no d.txt file in auth_handling.

```
$ git log --oneline
681d8dc (HEAD -> auth_handling) z.txt new file created in auth_handler is added.
f3e3dc1 y.txt new file created in auth_handler is added.
49a16e5 x.txt new file created in auth_handler is added.
c270e52 c.txt added.
148ce7e b.txt added.
9965ce1 a.txt added.
```

So, no commit which we committed for d.txt in master. (So all branch is isolated independent flow once branch is created 😊).

Note: In case of SVN for same operation we have to create a new directory and we have to copy all the files manually. So, it is very difficult and time-consuming activity when compare with other tools.

Que:

1. Is Git Create a new directory?
2. Does git copy all the files from parent to child?

Ans: No sir, Git won't create a new directory and won't copy all files.
Internally It is logical duplication of files

(To test it go to git folder ***D:\Git Project\branching_project*** and if we change branch accordingly that directory data will also be change 😊). So based on our branch files will be available in same folder.

- -

Important Notes:

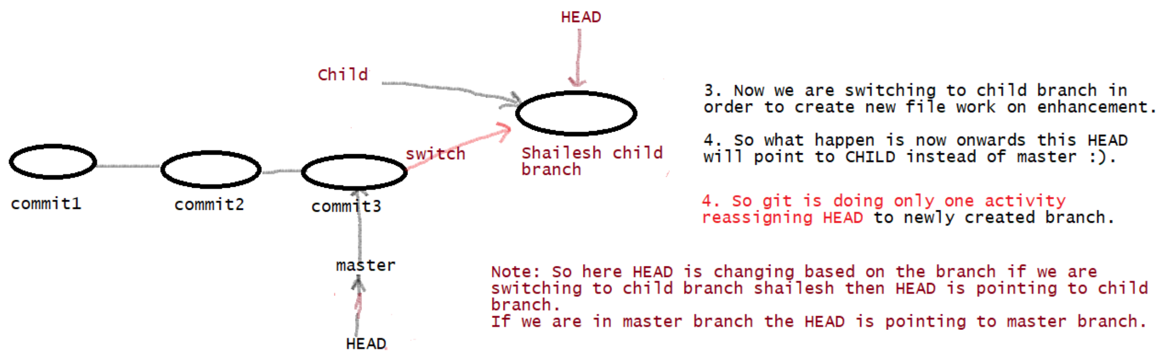
```
$ git checkout master
Switched to branch 'master'
```

Now the HEAD is pointing to master latest commit.

```
/D/Git Project/branching_project/.git (GIT_DIR!)
$ cat HEAD
ref: refs/heads/master
```

let me create a Shailesh branch from master

```
$ git branch Shailesh
/D/Git Project/branching_project (master)
```




```
$ git checkout shailesh
Switched to branch 'shailesh'

$ cat HEAD
ref: refs/heads/shailesh
```

To verify we know that in ***.git/refs/heads*** we will have all branch file details so let see the details.

```
$ cat .git/refs/heads/master
2ca0c7184c615c16bad0817b78f47ede59c39994

//master is always pointing to the latest commit of master branch.


/D/Git Project/branching_project (auth_handling)
$ cat .git/refs/heads/auth_handling
681d8dc4c25c1abd0a0d6d3a7732aa50ec567e7b
```

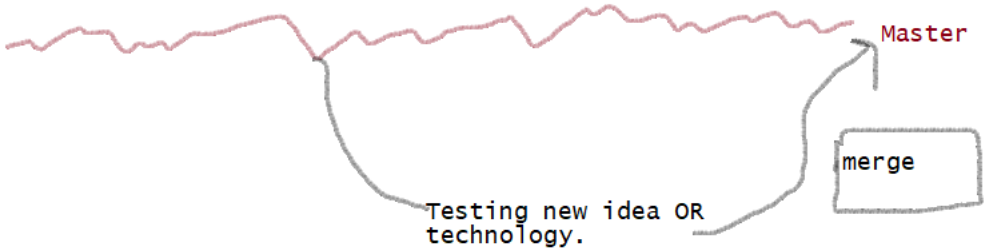
//above one is last/recent commit of ***auth_handling*** branch.

Note: So switching from one branch to another git basically It is just assigning HEAD refernce to latest commit of that branch 😊.

So git braching won't required any efforts so It is faster compare to other tools.

Chap3: Multiple use cases and Advantages of Branching.

1. We are required to work on new feature, but we don't know how much time it will take so better to create a new branch and once It is done then we can merge it.
2. hotfixes in production (a patch kind of thing).
3. To support multiple versions of same code base. (For every version, a separate branch will be there. If we want to fix any bugs or performance issues OR any changes in a particular version).
4. To Test new Ideas OR new Technologies without effecting main code base happily we can for branching concept.



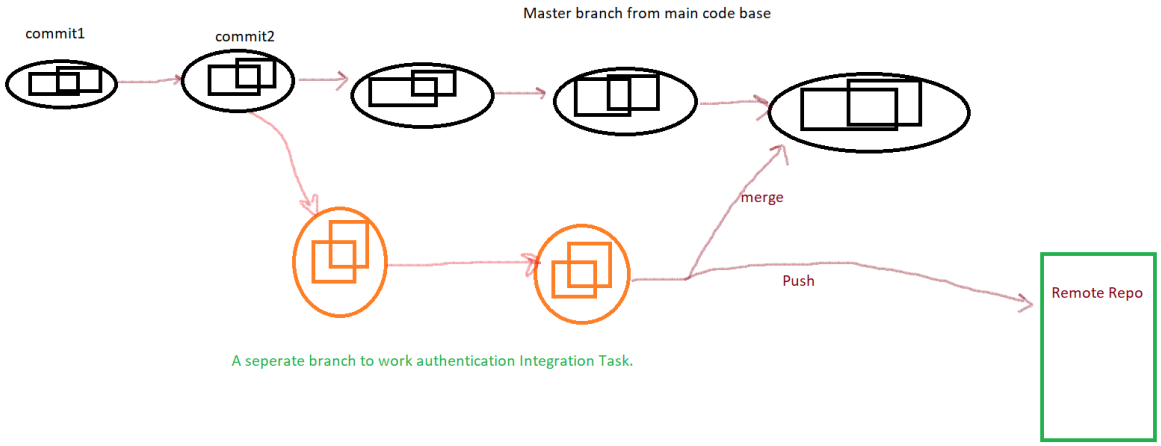
Advantages of branching:

1. Parallel development
2. We can work on multiple flows in isolated way.
3. We can organize source code in clean way.
4. Implementing new feature will become very easy.
5. Bug fixing will become very easy.
6. Testing new ideas OR new Technologies will become very easy.

Chap4:Fast forward merge and 3 Way merge.

Merging of a branch:

Suppose we have created a branch to implement a new feature. We completed Implementation off that new feature.
We must combine those changes from child base class with parent branch and then we will push that parent (master branch) to remote repository. This combining process is nothing but merging 😊.



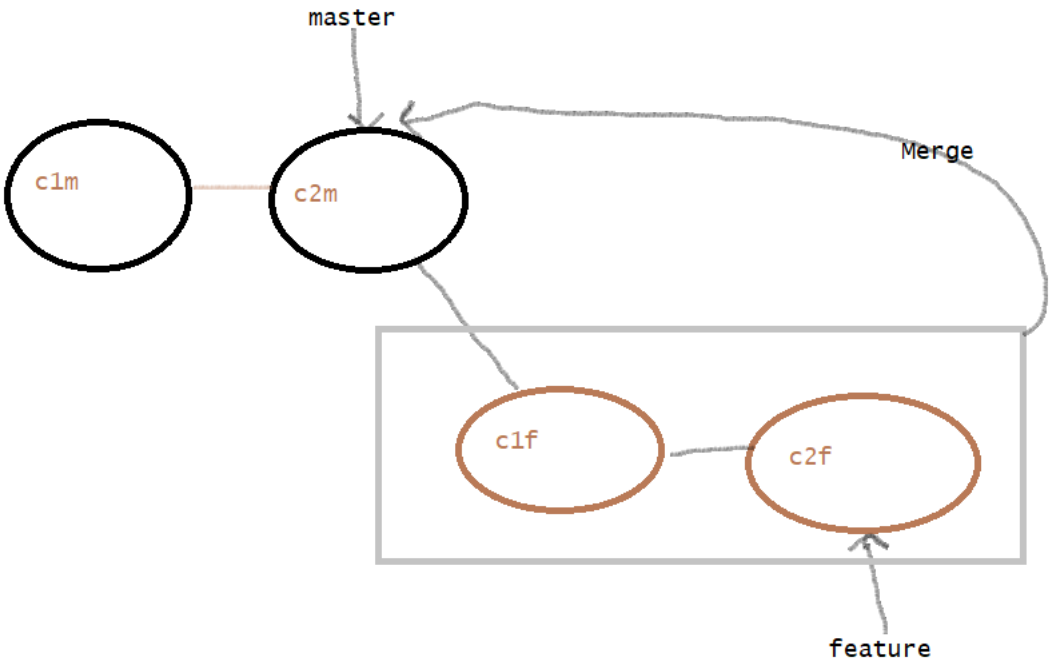
Sir I want to merge child class changes to master so from master class we need to use.

$git merge child_branch_name //from Master class (Because master class required this changes 😊).

Task:
In master branch I will create 2 commit and then create a feature (child branch) and then create another 2 commits as a part of enhancement and then merge it back with master 😊.

```
$ mkdir master_merge

$ cd master_merge/

/D/Git Project/master_merge (master)
$ git init
Initialized empty Git repository in D:/Git Project/master_merge/.git/

$ touch a.txt b.txt

/D/Git Project/master_merge (master)
$ git add a.txt; git commit -m "c1m-commit1 in master branch"
[master (root-commit) 8cd495a] c1m-commit1 in master branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt

/D/Git Project/master_merge (master)
$ git add b.txt; git commit -m "c2m-commit2 in master branch"
[master da7564a] c2m-commit2 in master branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 b.txt
```

So total 2 commits are there in the master:




```
/D/Git Project/master_merge (master)
$ git log --oneline
da7564a (HEAD -> master) c2m-commit2 in master branch
8cd495a c1m-commit1 in master branch
```

And branches:

```
/D/Git Project/master_merge (master)
$ git branch
* master
```

Now we will create a new branch with name as feature and switch to it. (In single command by using 😊).

```
$ git checkout -b feature
Switched to a new branch 'feature'

/D/Git Project/master_merge (feature)
$ git branch
* feature
  master
```

Now let us create 2 files in feature branch.

```
$ touch x.txt y.txt

/D/Git Project/master_merge (feature)
$ git add x.txt;git commit -m "c1-f- commit1 in feature branch.(Added new feature)"
[feature 3f0a6ac] c1-f- commit1 in feature branch.(Added new feature)
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 x.txt

/D/Git Project/master_merge (feature)
$ git add y.txt;git commit -m "c2-f- commit2 in feature branch.(Added new feature)"
[feature 4554020] c2-f- commit2 in feature branch.(Added new feature)
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 y.txt
```

```
/D/Git Project/master_merge (feature)
$ git log --oneline
4554020 (HEAD -> feature) c2-f- commit2 in feature branch.(Added new feature)
3f0a6ac c1-f- commit1 in feature branch.(Added new feature)
da7564a (master) c2m-commit2 in master branch
8cd495a c1m-commit1 in master branch
```

We know here we have switched to feature branch, so HEAD is pointing to feature branch.

```
/D/Git Project/master_merge (feature)
$ ls
a.txt  b.txt  x.txt  y.txt
```

// a.txt, b.txt is coming from parent.

Now I want to merge changes done from my feature branch to master branch. So, we need to execute ***$git merge branch_name*** from master branch.

```
/D/Git Project/master_merge (feature)
$ git checkout master
Switched to branch 'master'

/D/Git Project/master_merge (master)
$ git branch

//Git merging

/D/Git Project/master_merge (master)
$ git merge feature
Updating da7564a..4554020
Fast-forward
 x.txt | 0
 y.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 x.txt
 create mode 100644 y.txt
```

Now let us verify

```
/D/Git Project/master_merge (master)
$ git branch
  feature
* master

/D/Git Project/master_merge (master)
$ ls
a.txt  b.txt  x.txt  y.txt

/D/Git Project/master_merge (master)          //staging
$ git ls-files
a.txt
b.txt
x.txt
y.txt

/D/Git Project/master_merge (master)
$ git log --oneline
4554020 (HEAD -> master, feature) c2-f- commit2 in feature branch.(Added new
feature)
3f0a6ac c1-f- commit1 in feature branch.(Added new feature)
da7564a c2m-commit2 in master branch
8cd495a c1m-commit1 in master branch
```

Note: After merging feature branch will exist but if we want, we can delete because changes are already merged to master branch.
Proof:

```
/D/Git Project/master_merge (master)
$ git branch
  feature
* master
```

In above case git performed Fast-forward merge. So, question what is Fast-forward merge and what are the Types?

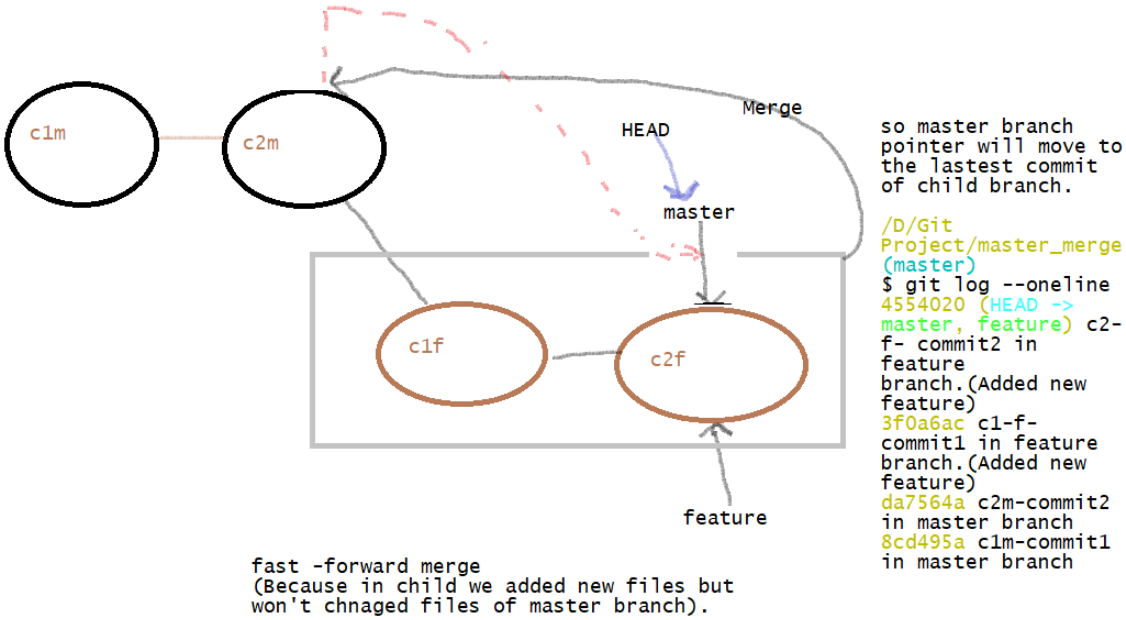
There are two types of merges:

Fast-Forward merge	Three-way merge

a) Fast-forward merge:

After creating child branch if we are not doing any changes in parent branch files (which came from parent) then git will perform fast-forward merge. (Parent branch files won't change/master branch unchanged).

In the fast-forward merge, git simply move parent branch pointer to the last commit on child branch.



So master, feature and HEAD are pointing to the c2f latest commit of child after merge. (Now if we want we can delete complete feature branch).
The advantage is that the feature branch code is developed parallelly.

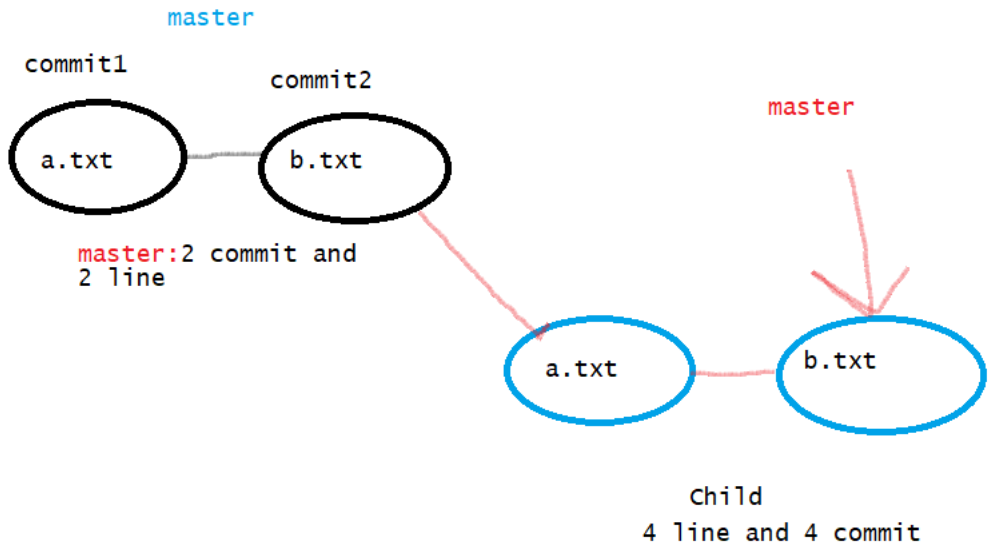
Note: Once it is merged and if required happily, we can create another branch from master for another enhancement work.

In fast-forward merge there is no chance of merge conflicts because up-dation/changes happen only in child branch not in parent branch.

E.g.) Don't think that if we'll modify same file say a.txt in its child branch then fast-forward will not happen 😊.

```
$ mkdir fast_forwatd_test
$ cd fast_forwatd_test/
$ git init
```

```
Initialized empty Git repository in D:/Git Project/fast_forwatd_test/.git/

$ echo "First line in a.txt -master branch" > a.txt

$ git add a.txt ; git commit -m "c1-m first commit in master "
warning: LF will be replaced by CRLF in a.txt.
The file will have its original line endings in your working directory
[master (root-commit) b523e27] c1-m first commit in master
 1 file changed, 1 insertion(+)
 create mode 100644 a.txt

Now let's add second line in a.txt file.

$ echo "second line in a.txt -master branch" >> a.txt

$ git add a.txt ; git commit -m "c2-m second commit in master "
warning: LF will be replaced by CRLF in a.txt.
The file will have its original line endings in your working directory
[master 2477187] c2-m second commit in master
 1 file changed, 1 insertion(+)


$ git log --oneline
2477187 (HEAD -> master) c2-m second commit in master
b523e27 c1-m first commit in master
```



Now we will create branch and check.

```
/D/Git Project/fast_forwatd_test (master)
$ git checkout -b feature
Switched to a new branch 'feature'

/D/Git Project/fast_forwatd_test (feature)
$ echo "third line in a.txt -feature branch" >> a.txt

/D/Git Project/fast_forwatd_test (feature)
$ git add a.txt ; git commit -m "f1-m first commit in feature"
warning: LF will be replaced by CRLF in a.txt.
The file will have its original line endings in your working directory
[feature 732683e] f1-m first commit in feature
 1 file changed, 1 insertion(+)

/D/Git Project/fast_forwatd_test (feature)
$ echo "4th line in a.txt -feature branch" >> a.txt

/D/Git Project/fast_forwatd_test (feature)
```

```
$ git add a.txt ; git commit -m "f1-m second commit in feature"
warning: LF will be replaced by CRLF in a.txt.
The file will have its original line endings in your working directory
[feature 69459d2] f1-m second commit in feature
 1 file changed, 1 insertion(+)
```

So now in feature branch we are having 4 commit and a.txt contain 4 lines.

```
/D/Git Project/fast_forwatd_test (feature)
$ cat a.txt
First line in a.txt -master branch
second line in a.txt -master branch
third line in a.txt -feature branch
4th line in a.txt -feature branch

/D/Git Project/fast_forwatd_test (feature)
$ git log --oneline
69459d2 (HEAD -> feature) f1-m second commit in feature
732683e f1-m first commit in feature
2477187 (master) c2-m second commit in master
b523e27 c1-m first commit in master
```

// now let us checkout to master branch.

```
/D/Git Project/fast_forwatd_test (feature)
$ git checkout master
Switched to branch 'master'

/D/Git Project/fast_forwatd_test (master)
$ cat a.txt
First line in a.txt -master branch
second line in a.txt -master branch
```

// in master we have only 2 commits.

```
/D/Git Project/fast_forwatd_test (master)
$ git log --oneline
2477187 (HEAD -> master) c2-m second commit in master
b523e27 c1-m first commit in master
```

//Now let us perform merge operation.

```
/D/Git Project/fast_forwatd_test (master)
$ git merge feature
Updating 2477187..69459d2
Fast-forward
 a.txt | 2 ++                              //2 lines added in a.txt
 1 file changed, 2 insertions(+)

/D/Git Project/fast_forwatd_test (master)
$ git log --oneline
69459d2 (HEAD -> master, feature) f1-m second commit in feature
732683e f1-m first commit in feature
2477187 c2-m second commit in master
b523e27 c1-m first commit in master
```

//Now we can see that master s pointing to feature latest commit 😊.

Now our master a.txt will contain 4 lines.

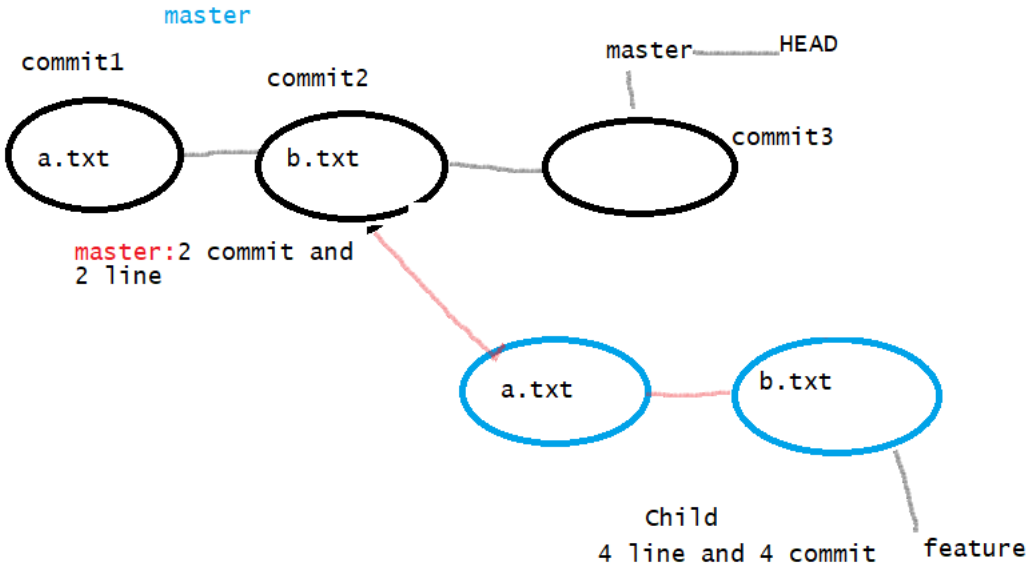
```
/D/Git Project/fast_forwatd_test (master)
$ cat a.txt
First line in a.txt -master branch
second line in a.txt -master branch
third line in a.txt -feature branch
4th line in a.txt -feature branch
```

So, we concluded that in fast forward there will be no conflict (because all changes happened in child feature branch and not in the master branch file).

What if master file also updated post to child branch creation and child branch also updated?
In this case 3-way merge will perform and there may be conflict and that we need to resolve.
After creating child branch if parent branch also contains some new commits, 3-way merge we need to use.

May be a chance of conflict.

a) **Without conflict:**




```
/D/Git Project (master)
$ mkdir merge_without_conflict

/D/Git Project (master)
$ cd merge_without_conflict/

/D/Git Project/merge_without_conflict (master)
$ git init
Initialized empty Git repository in D:/Git Project/merge_without_conflict/.git/

$ touch a.txt b.txt

/D/Git Project/merge_without_conflict (master)
$ git add a.txt;git commit -m "c1-m first commit in master repo."
[master (root-commit) a3adfbf] c1-m first commit in master repo.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt

/D/Git Project/merge_without_conflict (master)
$ git add b.txt;git commit -m "c2-m second commit in master repo."
[master 376431d] c2-m second commit in master repo.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 b.txt
```

so, in master 2 files and 2 commit.

```
/D/Git Project/merge_without_conflict (master)
$ git log --oneline
376431d (HEAD -> master) c2-m second commit in master repo.
a3adfbf c1-m first commit in master repo.
```

Now let create child branch and create another file.

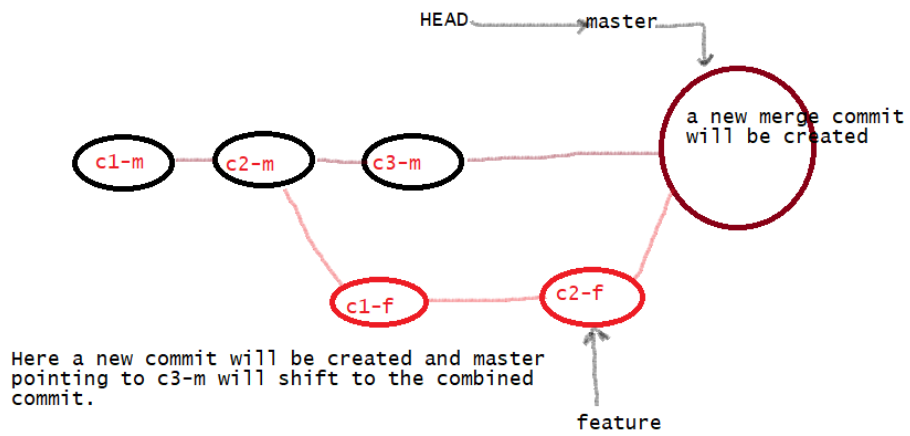
```
$ git checkout -b feature
Switched to a new branch 'feature'

$ touch x.txt y.txt

/D/Git Project/merge_without_conflict (feature)
$ git add x.txt;git commit -m "f1-m first commit in feature repo."
[feature 6f408ab] f1-m first commit in feature repo.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 x.txt

/D/Git Project/merge_without_conflict (feature)
$ git add y.txt;git commit -m "f2-m second commit in feature repo."
[feature c74dfe5] f2-m second commit in feature repo.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 y.txt
```

So now in child class also 2 commits created total=4.

```
/D/Git Project/merge_without_conflict (feature)
$ git log --oneline
c74dfe5 (HEAD -> feature) f2-m second commit in feature repo.
6f408ab f1-m first commit in feature repo.
376431d (master) c2-m second commit in master repo.
a3adfbf c1-m first commit in master repo.
```

//switch to master and create new file and commit as per the dig

```
$ git checkout master
Switched to branch 'master'

/D/Git Project/merge_without_conflict (master)
$ touch c.txt;git add c.txt;git commit -m "c3-m 3rd commit in master repo"
[master 89836eb] c3-m 3rd commit in master repo
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c.txt

/D/Git Project/merge_without_conflict (master)
$ ls
a.txt  b.txt  c.txt

/D/Git Project/merge_without_conflict (master)
$ git log --oneline
89836eb (HEAD -> master) c3-m 3rd commit in master repo
376431d c2-m second commit in master repo.
a3adfbf c1-m first commit in master repo.
```



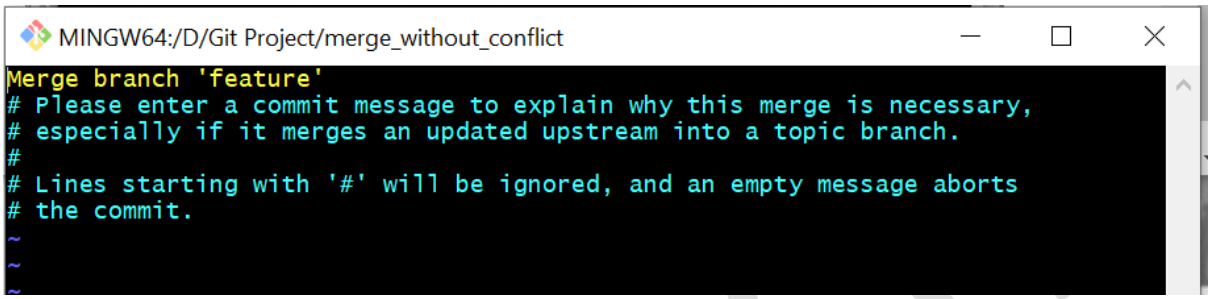
Both parent and child branches are updated so git perform 3-way merge 😊.

In fast forward no extra/new commit created but in case of 3 way merge a new commit will be created and pointer is pointing to new commit.
Let us merge feature branch.

Note: Here we know sir when we commit message we enter -m message and in-case if we are not doing that then we will get prompt message and It'll ask us to enter details.

Here also if we use $git merge feature basically It is 3way merge so It will create a new commit and for same It will prompt for commit message 😊.

```
$ git merge feature
```



I am quitting :wq and going with default command. Now

```
/D/Git Project/merge_without_conflict (master)
$ git merge feature
Merge made by the 'ort' strategy.
 x.txt | 0
 y.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 x.txt
 create mode 100644 y.txt
```

Note: So as per the above diagram we have 3 commits in master and 2 commit in child(feature) but now after merging how many commits will be there?
Ans: 6 😊 (Because new merge commit Added).

```
/D/Git Project/merge_without_conflict (master)
$ git log --oneline
eb92e5c (HEAD -> master) Merge branch 'feature'      //default commit message
89836eb c3-m 3rd commit in master repo
c74dfe5 (feature) f2-m second commit in feature repo.
6f408ab f1-m first commit in feature repo.
376431d c2-m second commit in master repo.
a3adfbf c1-m first commit in master repo.
```

Note: In above case master files (a , b, c.txt) and child (x,y.txt) both are different so there is no conflict happen(if master and child both are modifying same file then conflict happen).

Note: By using ***--graph*** we can see the details.

```
/D/Git Project/merge_without_conflict (master)
$ git log --oneline --graph
*   eb92e5c (HEAD -> master) Merge branch 'feature'
|\
| * c74dfe5 (feature) f2-m second commit in feature repo.
| * 6f408ab f1-m first commit in feature repo.
* | 89836eb c3-m 3rd commit in master repo
|/
* 376431d c2-m second commit in master repo.
* a3adfbf c1-m first commit in master repo.
```

Note: For c3-m don't think why It's showing in child (see * it is part of master 😊).

Que: HEAD~! Is c3-m OR f2-m?

```
/D/Git Project/merge_without_conflict (master)
$ git show HEAD~1
commit 89836eb3cefc08424dd355416c20e0b58b36d101
Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>
Date:   Sun Feb 13 20:42:08 2022 +0530

    c3-m 3rd commit in master repo

diff --git a/c.txt b/c.txt
new file mode 100644
index 0000000..e69de29
```

Fast forward merge	3-way
1. After creating child only child branch is modified but parent branch is not modified then git will perform Fast forward merge. 2. There is no chance of any conflict. 3. Any new commit is not required/not created. 4. Everything is fully handled by git.	1. After creating child, both parent and child branches are updated then 3-way merge. 2. May be a chance of conflict. 3. A new commit will be created which is also known as merge commit. 4. If there will be any conflict, then first we need to resolve(manually) that and then git will allow merge operation.

Note: Alternative of merge is re-base but understanding of above concept is very Important.

Chap4:Merge conflicts and Resolution Process.

So, we discuss that git perform fast-forward and 3-way merge 😊.
After creating the branches, if updates are happened in both branches – 3way merge.

If the same file modified by both parent and child branches, then git halt/stop merging process and raise conflict message and that we need to resolve manually 😊.

Git will markup both branches in the file to resolve the conflict very easily.

We have to edit the file manually to finalize content.
And then we must add to the staging area and then we have to perform commit.

What if we have multiple files and large code, resolving conflict manually will take more time?

- We will see how we can use our configured p4merge tool.

Practical:

```
/D/Git Project (master)
$ mkdir merge-conflict_resolution_process

/D/Git Project (master)
$ cd merge-conflict_resolution_process/

$ git init
Initialized empty Git repository in D:/Git Project/merge-
conflict_resolution_process/.git/

$ echo "First line Added in a.txt master." > a.txt

/d/Git Project/merge-conflict_resolution_process (master)
$ git add a.txt;git commit -m "c1m-commit1 in master."
warning: LF will be replaced by CRLF in a.txt.
The file will have its original line endings in your working directory
[master (root-commit) f648ced] c1m-commit1 in master.
 1 file changed, 1 insertion(+)
 create mode 100644 a.txt
```

//first commit completed. Let me add new one.

```
/d/Git Project/merge-conflict_resolution_process (master)
$ echo "second line in a.txt" >> a.txt
```

//now let me commit it.

```
/d/Git Project/merge-conflict_resolution_process (master)
$ git add a.txt;git commit -m "c2m-commit2 in master."
warning: LF will be replaced by CRLF in a.txt.
The file will have its original line endings in your working directory
[master 8f5c00c] c2m-commit2 in master.
 1 file changed, 1 insertion(+)
```

Now we are having two commits.

```
/d/Git Project/merge-conflict_resolution_process (master)
$ git log --oneline
8f5c00c (HEAD -> master) c2m-commit2 in master.
f648ced c1m-commit1 in master.
```

Note: Alright Sir now in master we have 2 commits so let us create child branch and switch to it.

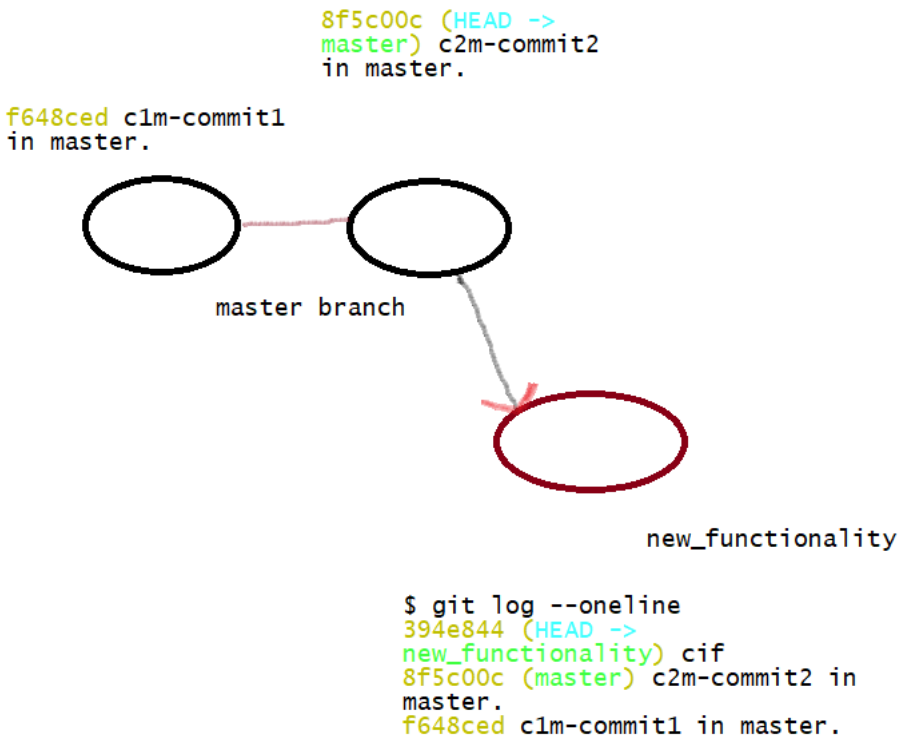
```
/d/Git Project/merge-conflict_resolution_process (master)
```

```
$ git checkout -b new_functionality
Switched to a new branch 'new_functionality'
```

//now in feature how many files and commit's will be available?
-Sir we know All the files and commit will be available from parent.

```
/d/Git Project/merge-conflict_resolution_process (new_functionality)
$ git log --oneline
8f5c00c (HEAD -> new_functionality, master) c2m-commit2 in master.
f648ced c1m-commit1 in master.
```

//Now let me add new data to new_functionality branch.



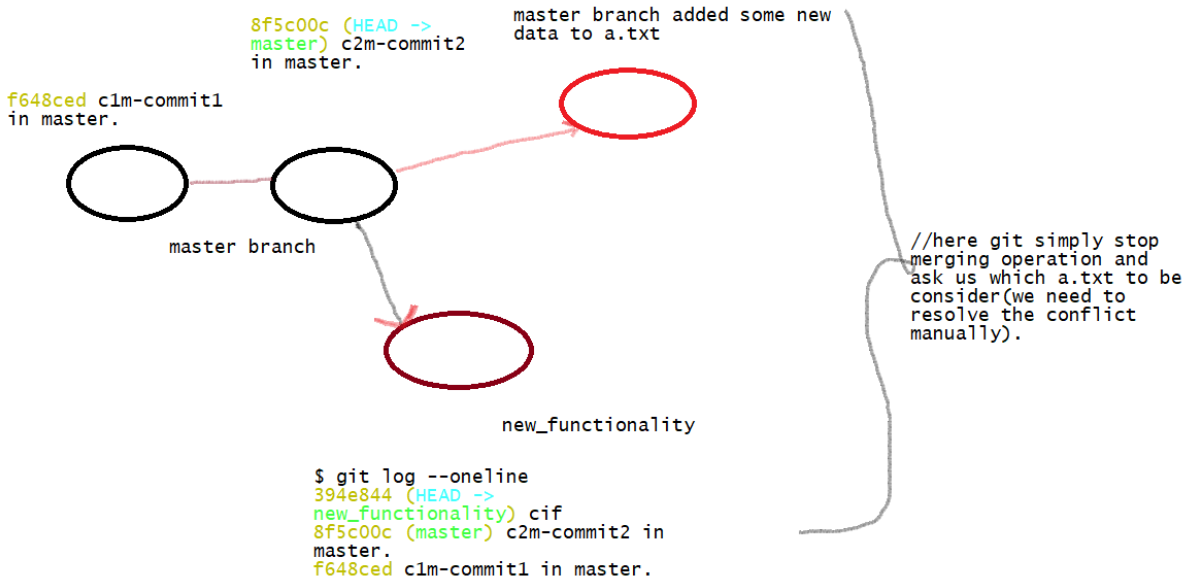

```
/d/Git Project/merge-conflict_resolution_process (new_functionality)
$ echo "New Data Added By Feature Branch" >> a.txt

/d/Git Project/merge-conflict_resolution_process (new_functionality)
$ git add a.txt; git commit -m "cif"
warning: LF will be replaced by CRLF in a.txt.
The file will have its original line endings in your working directory
[new_functionality 394e844] cif
 1 file changed, 1 insertion(+)

/d/Git Project/merge-conflict_resolution_process (new_functionality)
$ cat a.txt
First line Added in a.txt master.
second line in a.txt
New Data Added By Feature Branch
```

Now for conflict again we need to go into master branch and need to change content of a.txt same file.

```
$ git checkout master
Switched to branch 'master'
```

In master a.txt content is (only two lines because we didn't perform merged operation 😊).

```
$ cat a.txt
First line Added in a.txt master.
second line in a.txt

/d/Git Project/merge-conflict_resolution_process (master)
$ echo "New data Added by master branch." >> a.txt

/d/Git Project/merge-conflict_resolution_process (master)
$ git add a.txt ; git commit -m "c3m- commit no 3 from master branch"
warning: LF will be replaced by CRLF in a.txt.
The file will have its original line endings in your working directory
[master 7e90146] c3m- commit no 3 from master branch
 1 file changed, 1 insertion(+)
```



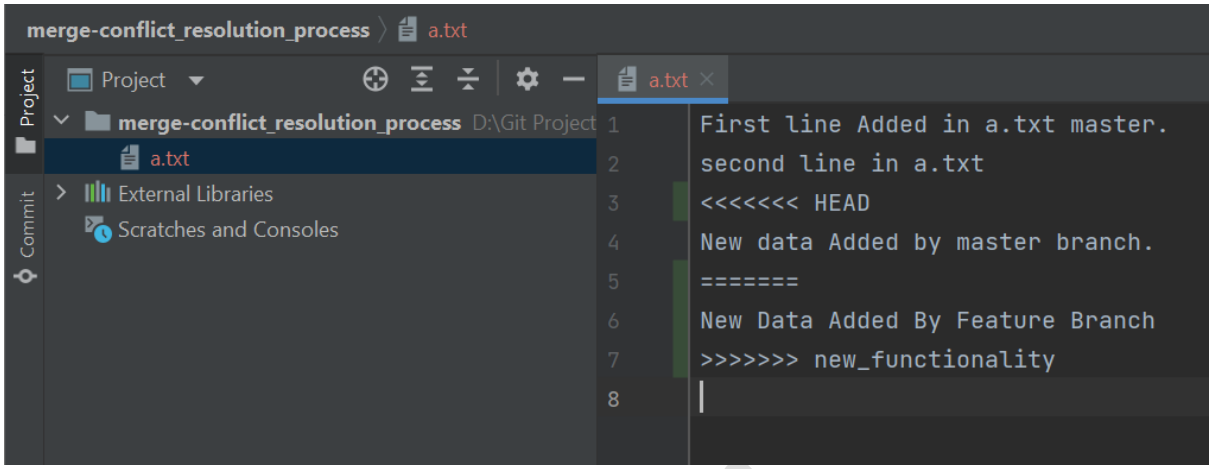
/d/Git Project/merge-conflict_resolution_process (master) $ git log --oneline 7e90146 (HEAD -> master) c3m- commit no 3 from master branch 8f5c00c c2m-commit2 in master. f648ced c1m-commit1 in master.	/d/Git Project/merge-conflict_resolution_process (new_functionality) $ git log --oneline 394e844 (HEAD -> new_functionality) cif 8f5c00c c2m-commit2 in master. f648ced c1m-commit1 in master.

Now let us merge. (It'll be 3-Way merge)

```
/d/Git Project/merge-conflict_resolution_process (master)
$ git merge new_functionality
Auto-merging a.txt
CONFLICT (content): Merge conflict in a.txt
Automatic merge failed; fix conflicts and then commit the result.
```

//if I will use Pycharm or VS we can see the ERROR there in the file like.

D:\Git Project\merge-conflict_resolution_process> git merge new_functionality
error: Merging is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.



Note: If we will check commit, we can see that merge commit will not be there because it got failed.

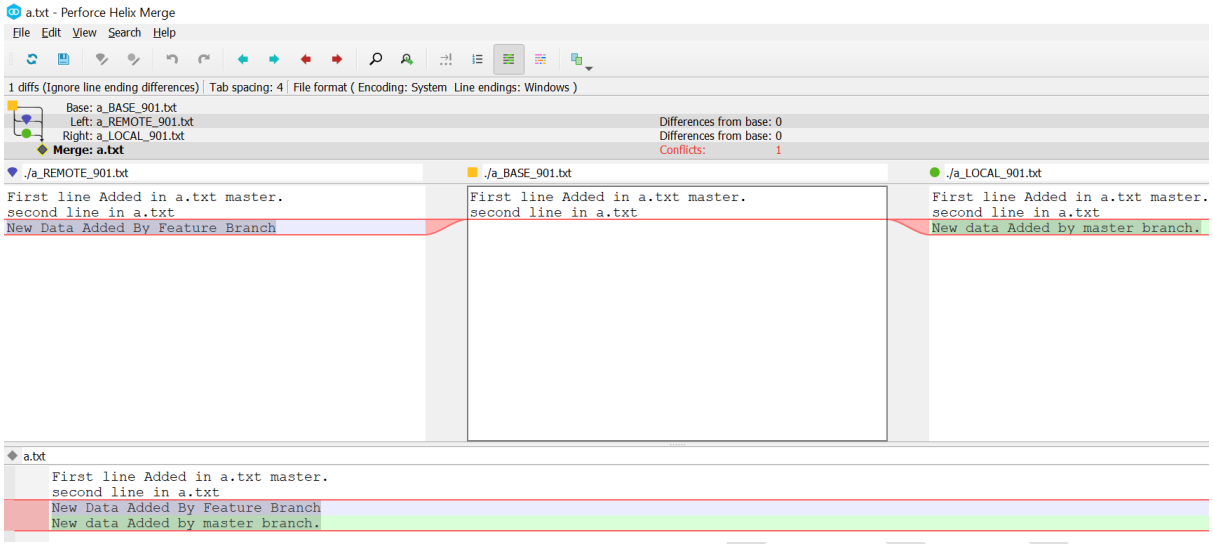
```
/d/Git Project/merge-conflict_resolution_process (master|MERGING)
$ git log --oneline
7e90146 (HEAD -> master) c3m- commit no 3 from master branch
8f5c00c c2m-commit2 in master.
f648ced c1m-commit1 in master.


$ 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:   a.txt

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

no changes added to commit (use "git add" and/or "git commit -a")
```

```
/d/Git Project/merge-conflict_resolution_process (master|MERGING)
$ cat a.txt
First line Added in a.txt master.
second line in a.txt
<<<<<<< HEAD
New data Added by master branch.
=======
New Data Added By Feature Branch
>>>>>>> new_functionality
```

//Now we have already configured git merged tool right.

```
/d/Git Project/merge-conflict_resolution_process (master|MERGING)
$ git config --global --list
user.name=shailesh
user.email=shaileshyadav7958@gmail.com
merge.tool=p4merge
merge.tool=p4merge.path
merge.tool=p4merge.path
merge.tool=p4merge.prompt
diff.tool=p4merge
diff.tool=p4merge.path
diff.tool=p4merge.prompt
alias.one=log --oneline
alias.s=status
alias.s1=git status
```

Just type
```
$ git mergetool
```



In above local means master branch. (We can see the no of conflict in above case now It is only one).
If multiple conflict, then we can check by clicking on below symbol.

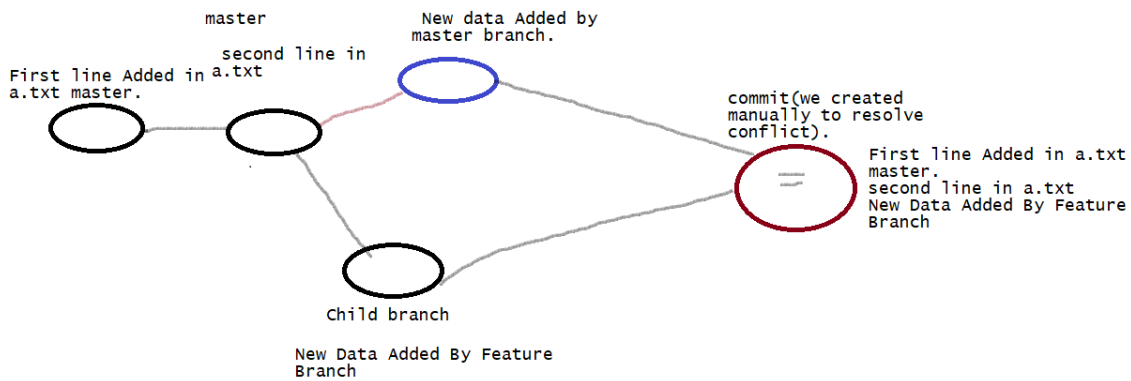
So, we need to select line which we want to save and then click on Save button.

```
/d/Git Project/merge-conflict_resolution_process (master|MERGING)
$ cat a.txt
First line Added in a.txt master.
second line in a.txt
New Data Added By Feature Branch
```

So here we have considered the feature branch commit 😊.
Note: If we want both files commit no problem, sir remove `<<<<<<< >>>>> ==== and save file.`

We have edited file and now we need to add it to staging and then we must perform commit.

```
$ git add .

s (master|MERGING)
$ git commit -m "Resolved Merge conflicts.."
[master fefda82] Resolved Merge conflicts..
```




```
/d/Git Project/merge-conflict_resolution_process (master)
```

```
$ git log --oneline --graph
*   fefda82 (HEAD -> master) Resolved Merge conflicts..
|\
| * 394e844 (new_functionality) cif
* | 7e90146 c3m- commit no 3 from master branch
|/
* 8f5c00c c2m-commit2 in master.
* f648ced c1m-commit1 in master.

/d/Git Project/merge-conflict_resolution_process (master)
$ git log --oneline
fefda82 (HEAD -> master) Resolved Merge conflicts..
7e90146 c3m- commit no 3 from master branch
394e844 (new_functionality) cif
8f5c00c c2m-commit2 in master.
f648ced c1m-commit1 in master.
```

//here now we are seeing feature branch merge new_functionality.

Chap5:How to delete branch.

Once merging is done then happily, we can delete it (There is no meaning of maintaining it).
Deletion of the branch is optional.
The main purpose is just to keep our repository clean 😊.

Command:

git branch -d branch_name

```
/d/Git Project/merge-conflict_resolution_process (master)
$ git branch
* master
  new_functionality

$ git branch -d new_functionality
Deleted branch new_functionality (was 394e844).
```

//now let us cross check branch (we'll have only one branch).

```
$ git branch
* master
```

Note: Only branch removed but commit will be as it's.

```
$ git log --oneline --graph
*   fefda82 (HEAD -> master) Resolved Merge conflicts..
|\
| * 394e844 cif
* | 7e90146 c3m- commit no 3 from master branch
|/
* 8f5c00c c2m-commit2 in master.
* f648ced c1m-commit1 in master.

/d/Git Project/merge-conflict_resolution_process (master)
$ cat a.txt
First line Added in a.txt master.
second line in a.txt
New Data Added By Feature Branch
```

Node: Once branch is merged then we can remove it doing same without merge dangerous 😊.

Chap6:Rebasing concept and demo example.

Sir we know we have already discussed,

Merge operation

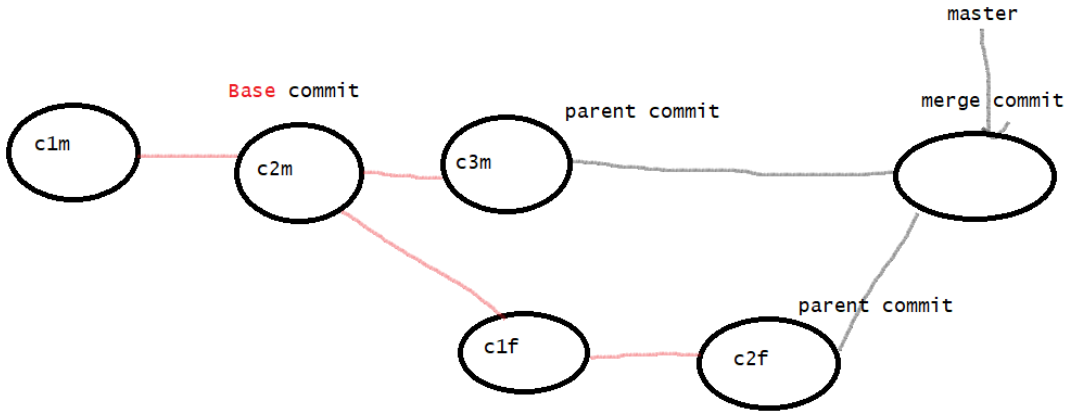
Fast-forward merge == > No conflict	3-way merge == > conflicts

And alternative to above concept we have rebase that we'll discuss in more details.

What is base commit?

- Sir it is the commit from which we are creating new child branch (checkout -b).

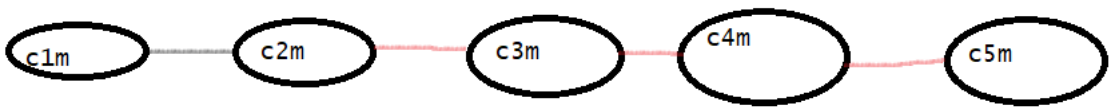
Example:



In case of merge-We can see a graph, which is may be non-linear (can see which commit is coming from child and which is from parent).
A new commit (merge commit) created.
May be a chance of merge conflicts.

My Requirement:

1. Linear flow instead of non-linear merge commit (easy for the programmer to track the changes).



2. New commit should not be created like merge commit (so we want only 5 commit instead of 6 compare with above fig.)
3. We should not have any conflicts.

So, we need some alternative of merging which solve/support above requirement then our life will become very simple 😊. And that is **rebase**.
Note: In case of rebase there will be no conflict.

Rebase is alternative to merge operation.
Rebase == > Re + Base === > Means rearrange base commit 😊.

Rebasing is two-step process whereas merge is single step process (git merge child-branch).

Process of rebasing:

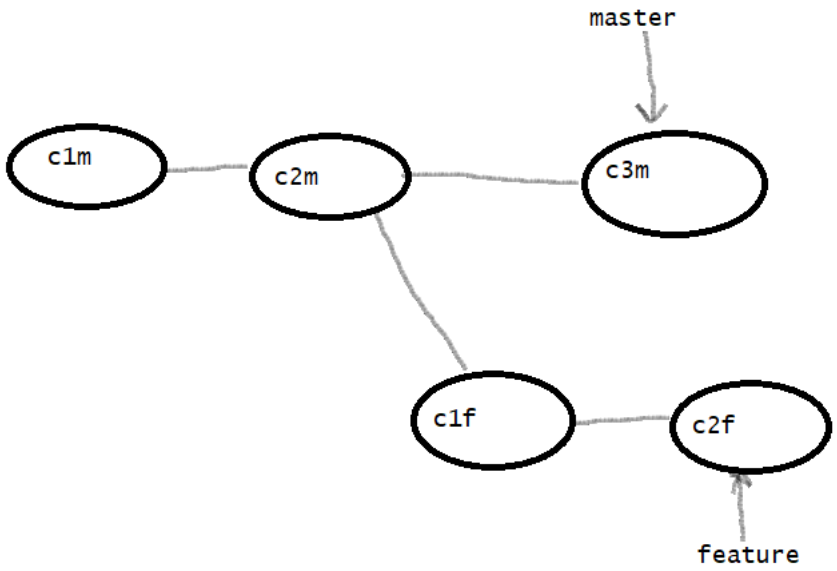
Step1: We have to rebase feature (child branch) branch on top of master branch.
Step2: We have to merge feature branch into master branch (Internally Fast-forward merge).

Now let us discuss each steps in details:

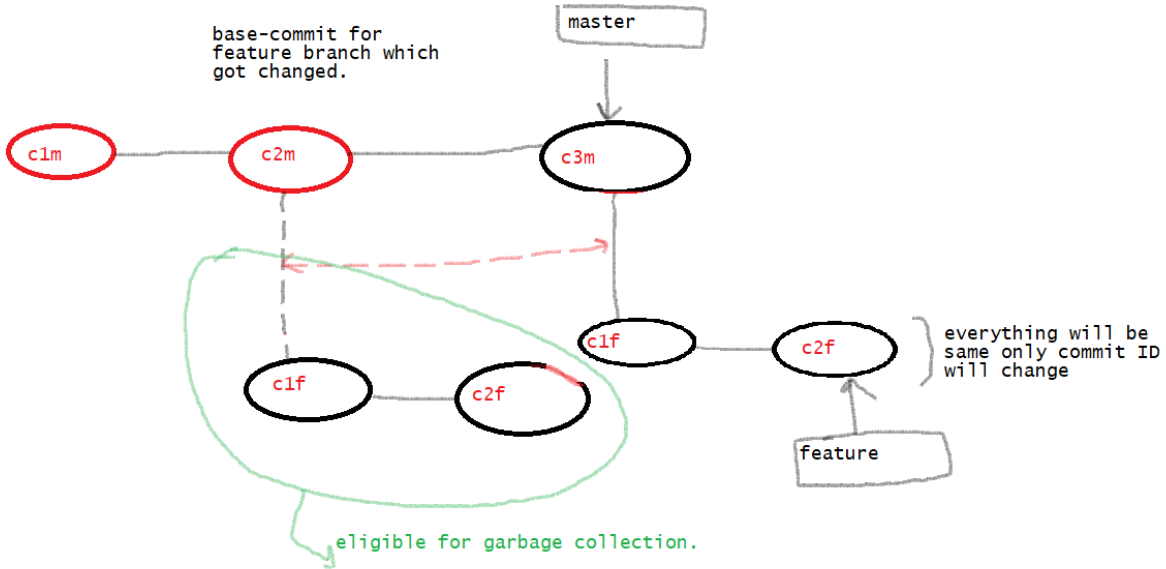
Step1: We have to rebase feature (child branch) branch on top of master branch.
//we need to run below two commands.

- git checkout feature/child-branch
- git rebase master

whatever new commits are there in the feature branch will be duplicated by git.
Here everything like commit message, author name, Mail ID will be same except commit ID's.



So in-case of rebase,

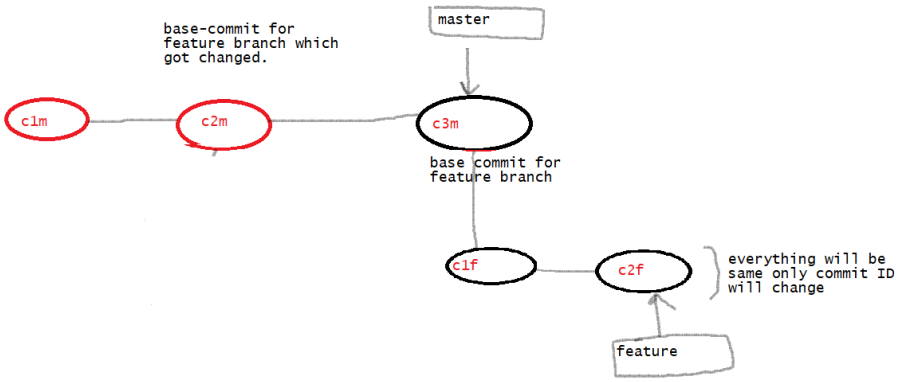


The base commit of feature branch will be updated as last commit of the parent branch.
Note: a duplicate of feature branch is created as shown in above diagram.

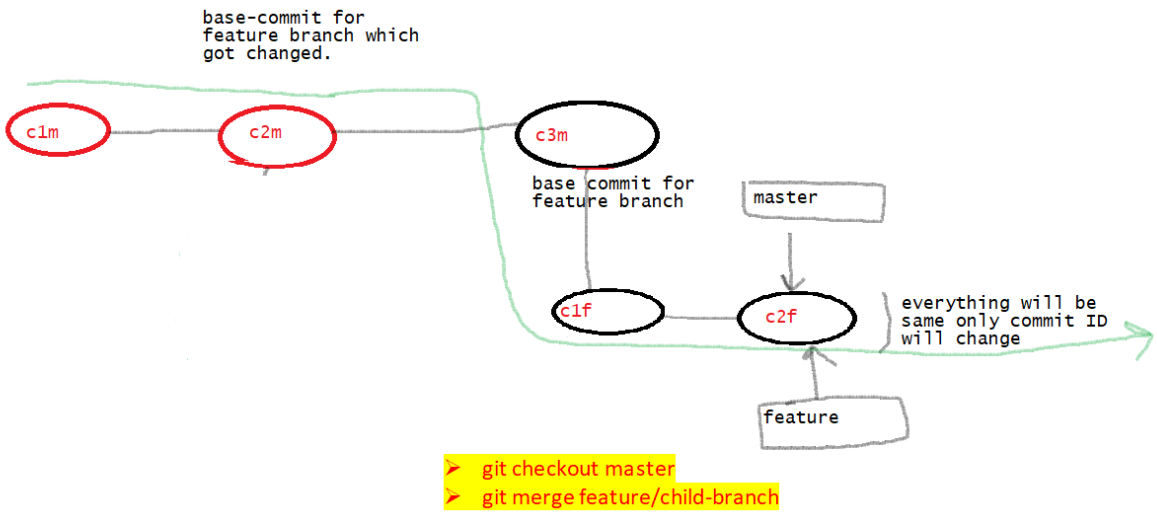
Step2: We have to merge feature branch into master branch (Internally Fast-forward merge).

- git checkout master
- git merge feature/child-branch

Note: Now after step1 removing garbage collection part now my diagram.



Now if you see above diagram after creating child branch there is no update/commit in master so fast forward merge will happen.
//so, after step2- fast forward merge master will point to c2f😊.



- git checkout master
- git merge feature/child-branch

So, advantage is linear flow (history will become linear) every commit will have parent commit.
(Here It is fast-forward merge so there is no question of conflict).

Practical:

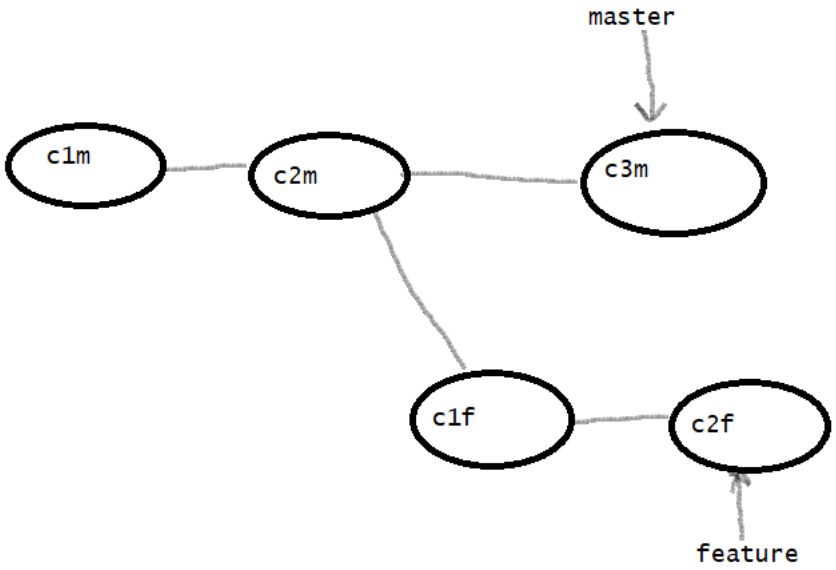
```
/d/Git Project (master)
$ mkdir rebasing

/d/Git Project (master)
$ cd rebasing/

/d/Git Project/rebasing (master)
$ git init
```

```
Initialized empty Git repository in D:/Git Project/rebasing/.git/

/d/Git Project/rebasing (master)
$ touch a.txt b.txt

/d/Git Project/rebasing (master)
$ git add a.txt ; git commit -m "c1m-commit1 in parent"
[master (root-commit) 53b37e3] c1m-commit1 in parent
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt

/d/Git Project/rebasing (master)
$ git add b.txt ; git commit -m "c2m-commit2 in parent"
[master 3a2c6dd] c2m-commit2 in parent
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 b.txt
```

so, we have total two commit in master branch

```
/d/Git Project/rebasing (master)
$ git log --oneline
3a2c6dd (HEAD -> master) c2m-commit2 in parent
53b37e3 c1m-commit1 in parent
```

Now we have completed master part of below diagram



Let us create child branch,

```
/d/Git Project/rebasing (master)
$ git checkout -b feature
Switched to a new branch 'feature'

/d/Git Project/rebasing (feature)
$ touch x.txt y.txt

/d/Git Project/rebasing (feature)
$ git add x.txt ; git commit -m "c1f-commit1 in feature branch"
[feature 6076049] c1f-commit1 in feature branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 x.txt

/d/Git Project/rebasing (feature)
$ git add y.txt ; git commit -m "c2f-commit2 in feature branch"
[feature ca08cef] c2f-commit2 in feature branch
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 y.txt

/d/Git Project/rebasing (feature)
```

```
$ git log --oneline
ca08cef (HEAD -> feature) c2f-commit2 in feature branch
6076049 c1f-commit1 in feature branch
3a2c6dd (master) c2m-commit2 in parent
53b37e3 c1m-commit1 in parent
```

Now we have created 2 commits in feature child branch. Let us create c3m in master.

```
$ git checkout master
Switched to branch 'master'

/d/Git Project/rebasing (master)
$ git log --oneline
3a2c6dd (HEAD -> master) c2m-commit2 in parent
53b37e3 c1m-commit1 in parent


/d/Git Project/rebasing (master)
$ touch c.txt

/d/Git Project/rebasing (master)
$ git add c.txt ; git commit -m "c3m-commit3 in parent"
[master 8f8b0fc] c3m-commit3 in parent
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c.txt
```

so now we are having 3 commits in master branch and 4 commit in child branch (2 inherited from parent/master).

---

step1: rebase feature branch on top of master branch
//let me check commit IDs before rebasing.

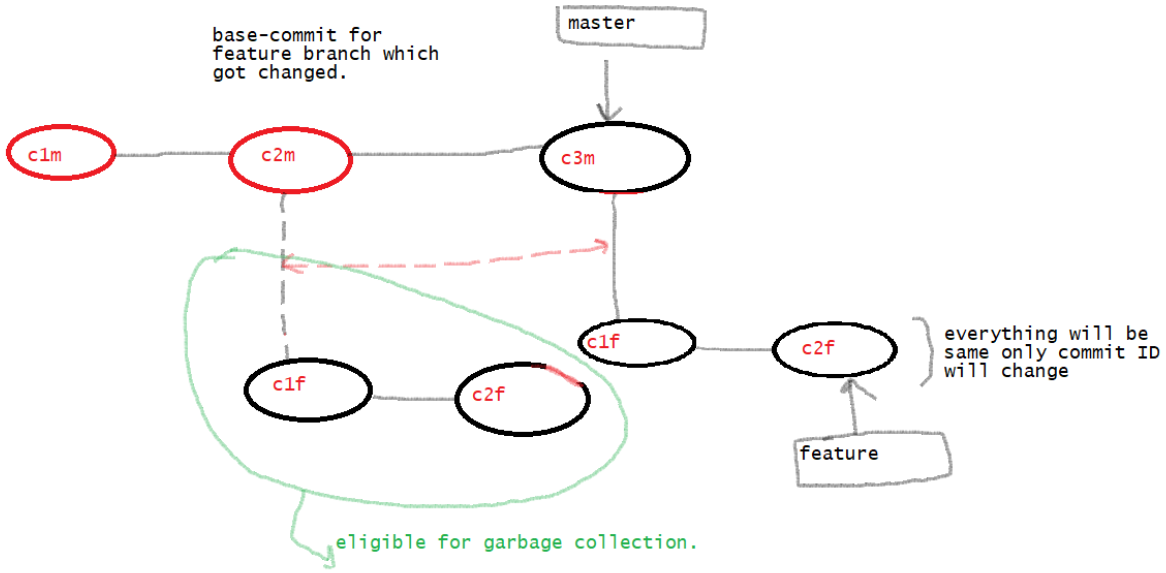
master	feature
/d/Git Project/rebasing (master) $ git log --oneline master 8f8b0fc (HEAD -> master) c3m-commit3 in parent 3a2c6dd c2m-commit2 in parent 53b37e3 c1m-commit1 in parent	/d/Git Project/rebasing (master) $ git log --oneline feature ca08cef (feature) c2f-commit2 in feature branch 6076049 c1f-commit1 in feature branch 3a2c6dd c2m-commit2 in parent 53b37e3 c1m-commit1 in parent

a) git checkout feature
b) git rebase master

//master to feature branch.

```
/d/Git Project/rebasing (master)
$ git checkout feature
Switched to branch 'feature'

/d/Git Project/rebasing (feature)
$ git rebase master
Successfully rebased and updated refs/heads/feature.
```

Now the diagram:



After using rebase we can see c1f and c2f will be removed and feature will point to c2f. Now if we check commit in feature branch then total 5 commits will be there.

```
/d/Git Project/rebasing (feature)
$ git log --oneline
782a34a (HEAD -> feature) c2f-commit2 in feature branch
e255529 c1f-commit1 in feature branch
8f8b0fc (master) c3m-commit3 in parent
3a2c6dd c2m-commit2 in parent
53b37e3 c1m-commit1 in parent
```

So, we can see that child commit ID are changed 😊.

```
/d/Git Project/rebasing (feature)
$ git log --oneline --graph
* 782a34a (HEAD -> feature) c2f-commit2 in feature branch
* e255529 c1f-commit1 in feature branch
* 8f8b0fc (master) c3m-commit3 in parent
* 3a2c6dd c2m-commit2 in parent
* 53b37e3 c1m-commit1 in parent
```

Note: before rebasing c3m is master branch commit and in child we had 4 commits (2 inherited from parent) but now in we can see after rebasing c3m also there.
Note: even if we check we will find that timestamp, author, email ID everything will be same if we use git log branch name.

Step2: Merge feature branch into master branch (fast forward merge).

```
/d/Git Project/rebasing (feature)
$ git checkout master
Switched to branch 'master'

/d/Git Project/rebasing (master)
$ git merge feature
Updating 8f8b0fc..782a34a
Fast-forward
 x.txt | 0
 y.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 x.txt
 create mode 100644 y.txt
```

Now whatever changes will be there in feature will be available to master branch.

```
/d/Git Project/rebasing (master)
```

```
$ ls
a.txt  b.txt  c.txt  x.txt  y.txt

/d/Git Project/rebasing (master)
$ git log --oneline
782a34a (HEAD -> master, feature) c2f-commit2 in feature branch
e255529 c1f-commit1 in feature branch
8f8b0fc c3m-commit3 in parent
3a2c6dd c2m-commit2 in parent
53b37e3 c1m-commit1 in parent
```

So, after merge we can see that master and feature are pointing to the same c2f commit.
Now if we check graph It'll be linear.

```
/d/Git Project/rebasing (master)
$ git log --oneline --graph
* 782a34a (HEAD -> master, feature) c2f-commit2 in feature branch
* e255529 c1f-commit1 in feature branch
* 8f8b0fc c3m-commit3 in parent
* 3a2c6dd c2m-commit2 in parent
* 53b37e3 c1m-commit1 in parent
```

Now work with feature branch completed so we can delete it.

```
/d/Git Project/rebasing (master)
$ git branch -d feature
Deleted branch feature (was 782a34a).
```

Chap7:Advantages and disadvantages of rebasing difference when compared with merge.

Advantages of Rebase:

1. Rebase keeps history clean/linear.
   Every commit has single parent. (In case of merging 3 Way there are multiple parents)
2. Clear workflow (Linear) will be there. Easy to understand for the developer.
3. Internally Fast Forward, so no chance of conflict.
4. No extra commit like merge commit.

Disadvantage:

1. It re writes history. We can't see history of commits from feature branch.
2. We do not aware which changes are coming from feature branch.

Note: rebase is very dangerous command and only recommended to use on public repository It is not suggested for private repository. (Because on public repo people are creating separate branch and working if we use rebase it will re write history and we cannot see history of child branch)

In our local merging one branch with another branch there we can use rebase concept.

Difference between merge and rebase?

merge	rebase
1. It is single step process. git checkout master git merge feature	1. It is 2-step process. git checkout feature git rebase master git checkout master git merge feature
2. Merge preserve history of all commits.	2. Rebase clear history of feature branch.
3. The commits can have more than one parent hence history is non-linear.	3. Here every commit is having single parent hence history is linear.
4. Conflict	4. No chance of conflict
5. Here we are aware which changes are coming from feature.	5. Cannot aware which changes be coming from feature branch.
6. We can use merge on public repo.	6. Not recommended on public repo.
7. Here we know which branch merge to which branch.	7. Here we don't know complete things. (which changes are coming from which person).

Use case: Suppose if we are having multiple branches which we created in our local and don't want outside people (remote) to see details about our branches then we can go with rebase.

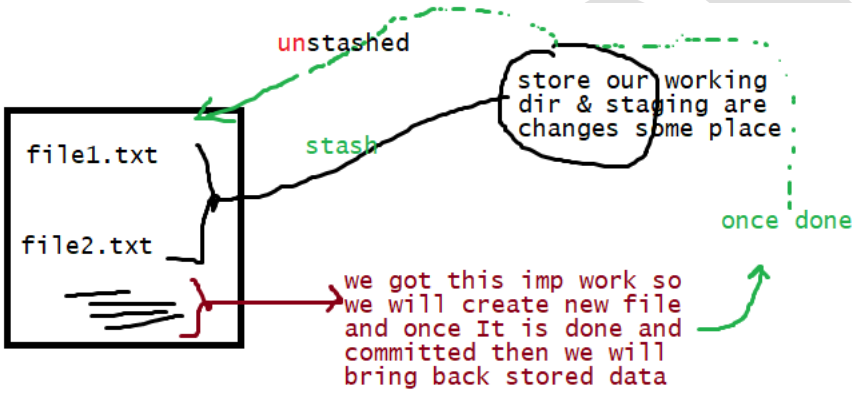
Chap8:git stash.

It is Advanced concept in GIT 😊.
So first what is the meaning of Stash?

Stash means storing safely in a hidden or secret place.

Real time example:
1) politicians OR rich person's 😊 They will store their money in Swiss Bank (A hidden/secret place).
2) Doctor he checks outside patient but in between if there will be any urgent case then he'll give priority to that and then continue with normal outside patient.

What is git Stash?
Ans: We have to save uncommitted changes from working directory and Staging area to some temporary location (due to this working tree will be clean) and then we can do our urgent work like Dr.:).
Once our work completed, we have to bring back the saved changes and then we can continue the work.



So, for storing our uncommitted changes we need to use **$ git stash command.**
For unstash – we need to use ==➔ git stash pop/apply

git stash => It is not applicable for newly created file. It is applicable for only tracked file.
To perform git stash at least one commit should be there.

Practical:

```
/D/Git Project (master)
$ mkdir stashing


/D/Git Project (master)
$ git init
Reinitialized existing Git repository in D:/Git Project/.git/

/D/Git Project/stashing (master)
$ echo "First line in file1" > file1.txt
/D/Git Project/stashing (master)
$ echo "First line in file2" > file2.txt
```

So till now both are untracked file.

```
$ git status
```

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt
```

Now can we use stash command now?
-No Sir, because as discussed two condition one is that it should have one initial commit and second It is applicable only for tracked file 😊.

So let add and commit.

```
/D/Git Project/stashing (master)
$ git add .;git commit -m "two files added."
The file will have its original line endings in your working directory
[master (root-commit) f020e5d] two files added.
 2 files changed, 2 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.txt

/D/Git Project/stashing (master)
$ git log --oneline
f020e5d (HEAD -> master) two files added.
```

Now we have one commit, and I got some work and for that I want to edit a.txt.

```
$ vi file1.txt
//added some data (working on it…)

/D/Git Project/stashing (master)
$ cat file1.txt
First line in file1
work is going on....
```

same way edits file2.txt and it's content will be

```
/D/Git Project/stashing (master)
$ cat file2.txt
First line in file2
work is in progress..
```

//now our working tree will not be clean because we did some changes after commit.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

//so, we can see above file is already tracked and modified.

```
/D/Git Project/stashing (master)
$ git add file2.txt

$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file2.txt
```

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
```

so uncommitted changes are there one is staging area other in working directory.

```
/D/Git Project/stashing (master)
$ git stash
Saved working directory and index state WIP on master: f020e5d two files added.
```

Now in above line working directory and index state means staging area – 2 files added.

Que: Now what will be the working tree?

Ans: It will be clean 😊,

```
/D/Git Project/stashing (master)
$ git status
On branch master
nothing to commit, working tree clean
```

que: Sir after git stash what will be the content of file1.txt?

Ans: It will show only one line because 2nd line (work is going on) was not committed.

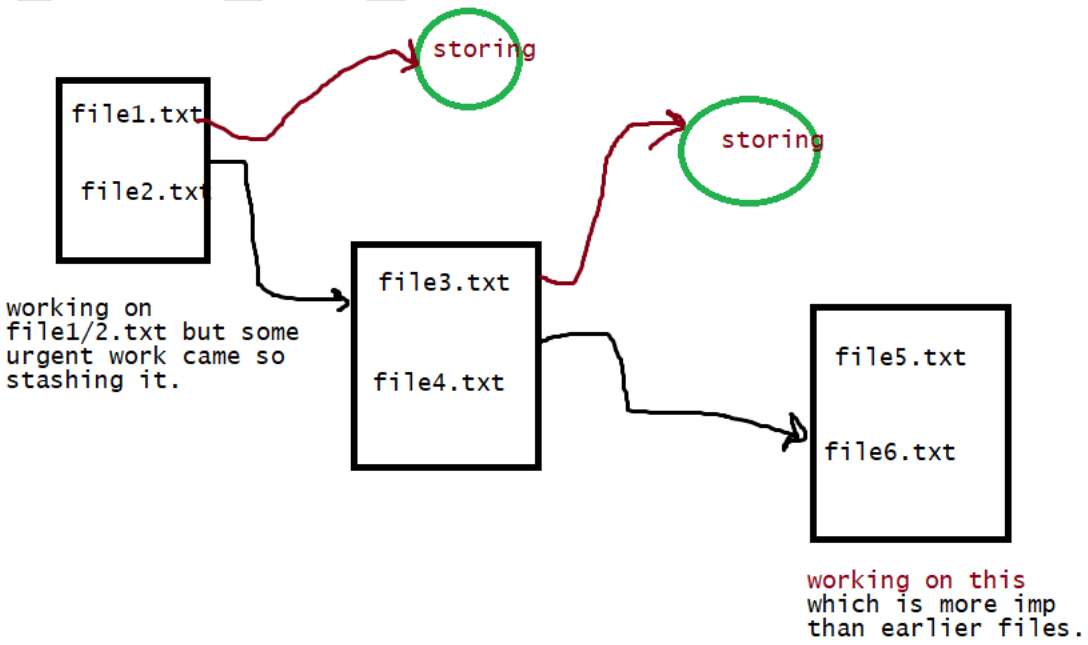
(For working tree to be clean It should not contain any modified are staging data so that is why it is removed from here 😊).

```
/D/Git Project/stashing (master)
$ cat file2.txt
First line in file2
```

Note: We can stash multiple time also. (Suppose a doctor is checking some emergency case but another case came which is even more urgent so his priority will change in same way suppose above we have used git stash once and working on some tasks, but some more urgent work came then no problem, sir we can stash it and work on more Imp work first 😊).

How to check all stashes are there?

By using **git stash list** command.



How many stashes we are having?

Ans :
```
/D/Git Project/stashing (master)
$ git stash list
stash@{0}: WIP on master: f020e5d two files added.
```

// So, we can see that here we are having only one stash 😊.

How to check content of it?

Ans: By using ***git show stash@{0}***

```
$ git show stash@{0}
commit 7dbf6ecffdfaf4ab38a728c3db36172ad8413f40 (refs/stash)
Merge: f020e5d 6d120b6
Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>
Date:   Thu [redacted]:21:57 2022 +0530

    WIP on master: f020e5d two files added.

diff --cc file1.txt
index 0bc9573,0bc9573..8563b20
--- a/file1.txt
+++ b/file1.txt
@@@ -1,1 -1,1 +1,2 @@@
  First line in file1
++work is going on....
```

So, we can see stashed line in green color.

Stash@{0} ➔ here 0 is stash ID.

Now our working tree will be clean so let me work on urgent work. 😊.

```
/D/Git Project/stashing (master)
$ git status
On branch master
nothing to commit, working tree clean

/D/Git Project/stashing (master)
$ echo "urgent work ...should be competed urgently " >> file3.txt

/D/Git Project/stashing (master)
$ cat file3.txt
urgent work ...should be competed urgently

/D/Git Project/stashing (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file3.txt

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

// So, now our git is talking only about text3.txt (not about which is already stashed).

```
/D/Git Project/stashing (master)
$ git add .;git commit -m "urgent work complted."
[master e45acca] urgent work complted.
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt
```

All right now we have completed our urgent work and now working tree will be clean.

```
/D/Git Project/stashing (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Let us now bring the stashed changes to working directory (By using *unstash* operation):

For this we have 2 ways'

1. git stash pop
2. git stash apply

```
/D/Git Project/stashing (master)
$ git stash list
stash@{0}: WIP on master: f020e5d two files added.

$ git stash pop stash@{0}
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped stash@{0} (7dbf6ecffdfaf4ab38a728c3db36172ad8413f40)
```

Now we can see we are getting earlier changes and working tree is not clean 😊.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt
```

Now let us check content of file1.txt

```
/D/Git Project/stashing (master)
$ cat file1.txt
First line in file1
work is going on....
```

now after this there will be no stash (because It'll remove entry from there).

```
/D/Git Project/stashing (master)
$ git stash list
```

git stash pop	Git stash apply
It will bring stashed changes from temporary location to working directory. The corresponding entry will be deleted.	It will bring stashed changes from temporary location to working directory. (Same thing) The corresponding entry won't be deleted. //so, if we required this stashed in any of other branch

Now I will work on my first/un stashed file and then let me add and commit it 😊.

```
/D/Git Project/stashing (master)
$ git add file1.txt file2.txt ; git commit -m "2 files added."
[master 79e412e] 2 files added.
 2 files changed, 2 insertions(+)


/D/Git Project/stashing (master)
```

```
$ git status
On branch master
nothing to commit, working tree clean
```

Note: After un stashed operation all the changes will be bring back to working directory.

```
/D/Git Project/stashing (master)
$ echo "Some more work is going ON ." >> file1.txt

/D/Git Project/stashing (master)
$ echo "Some more work is going ON ." >> file2.txt

$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt

no changes added to commit (use "git add" and/or "git commit -a")


/D/Git Project/stashing (master)
$ cat file1.txt
First line in file1
work is going on....
Some more work is going ON .


/D/Git Project/stashing (master)
$ cat file2.txt
First line in file2
work is in progress..
Some more work is going ON .
```

Note: Now suppose we have some Important work to do so let us **stash** it.

```
/D/Git Project/stashing (master)
$ git stash
Saved working directory and index state WIP on master: 79e412e 2 files added.
```

// two files added in stash.

```
/D/Git Project/stashing (master)
$ git stash list
stash@{0}: WIP on master: 79e412e 2 files added.
```

Now working tree will be clean

```
/D/Git Project/stashing (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Note: Now we got some urgent work.

```
/D/Git Project/stashing (master)
$ echo "very urgent work" > file4.txt

$ git add file4.txt ; git commit -m "very urgent work."
[master 35d7c1d] very urgent work.
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt
```

now working tree is clean

```
$ git status
On branch master
nothing to commit, working tree clean

/D/Git Project/stashing (master)
$ cat file1.txt
First line in file1
work is going on....

$ cat file2.txt
First line in file2
work is in progress..
```

Now our Imp work completed, and we can see the above file content now let us un-stashed it with second apply method.

```
/D/Git Project/stashing (master)
$ git stash apply stash@{0}
On branch master
Changes not staged for commit:
        modified:   file1.txt
        modified:   file2.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

Now if we check file content, we can see changes came back.

```
/D/Git Project/stashing (master)
$ cat file1.txt
First line in file1
work is going on....
Some more work is going ON .

/D/Git Project/stashing (master)
$ cat file2.txt
First line in file2
work is in progress..
Some more work is going ON .
```

Now what about entry?
It will be there because we are using apply method (corresponding entry won't be deleted and we can use in in one of our repositories 😊).

```
/D/Git Project/stashing (master)
$ git stash list
stash@{0}: WIP on master: 79e412e 2 files added.
```

Now once done we commit ongoing less priority job.

```
/D/Git Project/stashing (master)
$ git add file1.txt file2.txt;git commit -m "this job is also complted."
[master 9859cc3] this job is also complted.
 2 files changed, 2 insertions(+)
```

Chap9:partial stash.

Sir we know before this we have discussed for complete stash basically It will copy all the uncommitted files in working directory and staging area to separate place.
But some time we don't want to copy/store all files so for that we can go with partial stashing.

Even multiple files are there but we want to stash only some files so for that we need to use Partial stash.

//for full stash/complete stash git stash	//for partial stash git stash -p It will ask file by file file1 -- y file2 -- y file3 -- n

Practical:

```
/D/Git Project (master)
$ mkdir partialstash

/D/Git Project (master)
$ cd partialstash/

/D/Git Project/partialstash (master)
$ git init
Initialized empty Git repository in D:/Git Project/partialstash/.git/
```

Note: To perform stash one commit at least required. Stashing is applicable only for tracked file.

```
/D/Git Project/partialstash (master)
$ echo "first line " > file1.txt

/D/Git Project/partialstash (master)
$ echo "first line " > file2.txt

$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt

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

//so, for untracked file stashing is not possible.

```
/D/Git Project/partialstash (master)
$ git add .;git commit -m "first commit"
[master (root-commit) 7d79d14] first commit
 2 files changed, 2 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.txt
```

//now we have committed so both files are tracked.

```
/D/Git Project/partialstash (master)
```

```
$ git status
On branch master
nothing to commit, working tree clean
```

Now let us work on some requirement (Adding/modifying some data in it)

```
/D/Git Project/partialstash (master)
$ echo "work is going on"  >> file1.txt

/D/Git Project/partialstash (master)
$ echo "work is going on"  >> file2.txt

$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt

no changes added to commit (use "git add" and/or "git commit -a")

/D/Git Project/partialstash (master)
$ cat file1.txt
first line                              #already committed
work is going on                           #work is in progress.

/D/Git Project/partialstash (master)
$ cat file2.txt
first line
work is going on
```

Now suppose some urgent work came and we need to work on file2.txt (existing) and file3.txt (new). So, our requirement is to stash only file1.txt uncommitted changes. (i.e., work is going on)

```
/D/Git Project/partialstash (master)
$ git stash -p
diff --git a/file1.txt b/file1.txt
index eed1113..e30e3e5 100644
--- a/file1.txt
+++ b/file1.txt
@@ -1 +1,2 @@
 first line
+work is going on
(1/1) Stash this hunk [y,n,q,a,d,e,?]? y
diff --git a/file2.txt b/file2.txt
index eed1113..e30e3e5 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1 +1,2 @@
 first line
+work is going on
(1/1) Stash this hunk [y,n,q,a,d,e,?]? n

Saved working directory and index state WIP on master: 7d79d14 first commit
```

//Now we have stashed only file1.txt.

```
/D/Git Project/partialstash (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Note: for file1.txt stashing is already done (Ideally It should not show this 😊May be Bug). //we should get only file2.txt.

```
/D/Git Project/partialstash (master)
$ git stash list
stash@{0}: WIP on master: 7d79d14 first commit
```

//to check content:

```
$ git show stash@{0}
commit 554322c44ffe9703a4ce3ca17eb9734bd3e43cfc (refs/stash)
Merge: 7d79d14 fc59550
Author: shaileshyadav7771 <shaileshyadav7==@gmail.com>
Date:   Fri Feb 25 14:31:06 2028 +0530

    WIP on master: 7d79d14 first commit

diff --cc file1.txt
index eed1113,eed1113..e30e3e5
--- a/file1.txt
+++ b/file1.txt
@@@ -1,1 -1,1 +1,2 @@@
  first line
++work is going on
```

File1.txt is having 2 lines but un-committed line is stashed so we can see only one line data.

```
/D/Git Project/partialstash (master)
$ cat file1.txt
first line
```

but file2.txt is having 2 lines.

```
/D/Git Project/partialstash (master)
$ cat file2.txt
first line
work is going on
```

So now we can continue to work on file2.txt and even create new files.

```
/D/Git Project/partialstash (master)
$ echo "some urgent work" > file3.txt

$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
        modified:   file2.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file3.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

now above file2.txt is modified (file1.txt showing wrongly) and file3.txt is untracked file.

```
/D/Git Project/partialstash (master)
$ git add .;git commit -m "commit for urgent work."
[master 04c2c6d] commit for urgent work.
 2 files changed, 2 insertions(+)
 create mode 100644 file3.txt
```

Note: Above if you can see that 2-file changed / 2 file insertions but file1.txt is not added. 😊.

```
/D/Git Project/partialstash (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Now let us bring back file1.txt untracked data. (So, for this we can use either apply OR pop command).

```
$ git stash pop stash@{0}
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped stash@{0} (554322c44ffe9703a4ce3ca17eb9734bd3e43cfc)
```

// Now we have brought back file1.txt so tree will not be clear.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Now file1.txt contain

```
/D/Git Project/partialstash (master)
$ cat file1.txt
first line
work is going on
```

Chap10:How to delete stash.

Note: if we use ***git stash pop stash@{0}*** then stash entry will be deleted but if we use ***git stash apply stash@{0}*** then entry won't be deleted.

So once our job is completed then we can delete all the stashes.
We can have multiple stashes. We can delete all stashes OR a particular stash based on our requirement.

git stash clear	git stash drop stach_ID
// To delete all available stashes.	//to delete a particular stash. e.g.: git stash drop stash@{0}#0,1,2,3, no's

```
/D/Git Project (master)
$ mkdir git_stash_deleting
```

```
/D/Git Project/git_stash_deleting (master)
$ git init
Initialized empty Git repository in D:/Git Project/git_stash_deleting/.git/

$ echo "First Line " > file1.txt

$ echo "First Line " > file2.txt

$ echo "First Line " > file3.txt
```

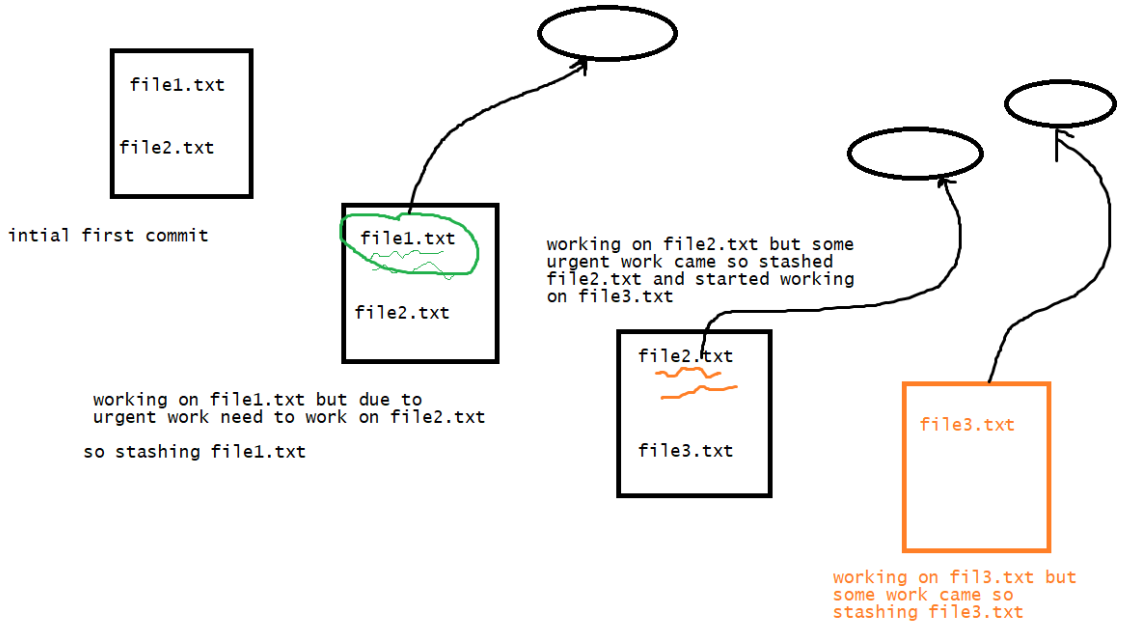
```
/D/Git Project/git_stash_deleting (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt
        file3.txt
nothing added to commit but untracked files present (use "git add" to track)
```

```
/D/Git Project/git_stash_deleting (master)
$ git add .;git commit -m "first commit"
[master (root-commit) 837b5d9] first commit
 3 files changed, 3 insertions(+)
 create mode 100644 file1.txt
 create mode 100644 file2.txt
 create mode 100644 file3.txt
```

Now at least one commit is there, and all 3 files are tracked. now suppose we are working on file1.txt and in between some imp work came 😊.

Clients want me to pause work on file1.txt (stashed it) and work on file2.txt.

Now if I will use ***$git stash*** then only one file file1.txt will be stashed because it is only modified/untracked.



So, we can see that working tree is clean.

```
/D/Git Project/git_stash_deleting (master)
$ git status
On branch master
nothing to commit, working tree clean
```

//now let me work on file1.txt

```
/D/Git Project/git_stash_deleting (master)
$ echo "work is going on .." >> file1.txt
```

//So, file1.txt is modified and we have not touched file2.txt and file3.txt.

```
/D/Git Project/git_stash_deleting (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file1.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

//now let me use git stash (in our case only file1.txt will be stashed).

```
/D/Git Project/git_stash_deleting (master)
$ git stash
Saved working directory and index state WIP on master: 837b5d9 first commit
```

//now our working directory will be clean because those uncommitted changes are stashed.

```
/D/Git Project/git_stash_deleting (master)
$ git status
On branch master
nothing to commit, working tree clean
```

//now to check total stash list

```
/D/Git Project/git_stash_deleting (master)
```

```
$ git stash list
stash@{0}: WIP on master: 837b5d9 first commit
```
Now I will start working on file2.txt

```
/D/Git Project/git_stash_deleting (master)
$ echo "work is going on" >> file2.txt

/D/Git Project/git_stash_deleting (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file2.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

// now again some more priority work came so let me stash it 😊.

```
/D/Git Project/git_stash_deleting (master)
$ git stash
Saved working directory and index state WIP on master: 837b5d9 first commit
```

//now again working tree will be clean.

```
/D/Git Project/git_stash_deleting (master)
$ git status
On branch master
nothing to commit, working tree clean
```

//so till now how many stashes we have?
Ans: 2 stashes

```
/D/Git Project/git_stash_deleting (master)
$ git stash list
stash@{0}: WIP on master: 837b5d9 first commit
stash@{1}: WIP on master: 837b5d9 first commit
```

//now let me start working on file3.txt

```
/D/Git Project/git_stash_deleting (master)
$ echo "work is going on .." >> file3.txt
```

Again git stash due to some Imp work 😊.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file3.txt

/D/Git Project/git_stash_deleting (master)
$ git stash
Saved working directory and index state WIP on master: 837b5d9 first commit
```

//Now total stashes –

```
/D/Git Project/git_stash_deleting (master)
$ git stash list
stash@{0}: WIP on master: 837b5d9 first commit
stash@{1}: WIP on master: 837b5d9 first commit
stash@{2}: WIP on master: 837b5d9 first commit
```

so, we can have multiple stashes.
Even we can check the details by using

```
$ git show stash stash@{0}
commit e281c8172b049b5be65ce3543f3dbb9638f4f77c (refs/stash)
Merge: 837b5d9 732ff06
Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>
Date:   [redacted]

    WIP on master: 837b5d9 first commit

diff --cc file3.txt
index 1fe1ebb,1fe1ebb..a65bb88
--- a/file3.txt
+++ b/file3.txt
@@@ -1,1 -1,1 +1,2 @@@
  First Line
++work is going on ..
```

Note: When we are using {0} It is showing for file3.txt 😊 May be last insert first O/p.

```
$ git show stash stash@{1}
commit e281c8172b049b5be65ce3543f3dbb9638f4f77c (refs/stash)
Merge: 837b5d9 732ff06
Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>
Date:   [redacted] +0530

    WIP on master: 837b5d9 first commit

diff --cc file3.txt
index 1fe1ebb,1fe1ebb..a65bb88
--- a/file3.txt
+++ b/file3.txt
@@@ -1,1 -1,1 +1,2 @@@
  First Line
++work is going on ..

commit 618bb6beb091b3f5f8de095f82225ed67824f058
Merge: 837b5d9 94eb0e1
Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>
Date:   [redacted] 2022 +0530

    WIP on master: 837b5d9 first commit

diff --cc file2.txt
index 1fe1ebb,1fe1ebb..49bcc02
--- a/file2.txt
+++ b/file2.txt
@@@ -1,1 -1,1 +1,2 @@@
  First Line
++work is going on
```

```
$ git show stash stash@{2}
commit e281c8172b049b5be65ce3543f3dbb9638f4f77c (refs/stash)
Merge: 837b5d9 732ff06
Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>
Date:   [redacted]

    WIP on master: 837b5d9 first commit

diff --cc file3.txt
index 1fe1ebb,1fe1ebb..a65bb88
--- a/file3.txt
+++ b/file3.txt
@@@ -1,1 -1,1 +1,2 @@@
  First Line
++work is going on ..

commit 5e0eb6decbc06c836fe3b3df6ba8e90473e87ef0
Merge: 837b5d9 0628ab7
Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>
Date:   Fri Feb 25 18:05:27 2022 +0530
```

```
    WIP on master: 837b5d9 first commit

diff --cc file1.txt
index 1fe1ebb,1fe1ebb..a65bb88
--- a/file1.txt
+++ b/file1.txt
@@@ -1,1 -1,1 +1,2 @@@
  First Line
++work is going on ..
```

Now my requirement is to delete a particular stash.

```
/D/Git Project/git_stash_deleting (master)
$ git stash drop stash@{2}
Dropped stash@{2} (5e0eb6decbc06c836fe3b3df6ba8e90473e87ef0)
```

Now let us verify it.

```
/D/Git Project/git_stash_deleting (master)
$ git stash list
stash@{0}: WIP on master: 837b5d9 first commit
stash@{1}: WIP on master: 837b5d9 first commit
```

//To delete all stashes

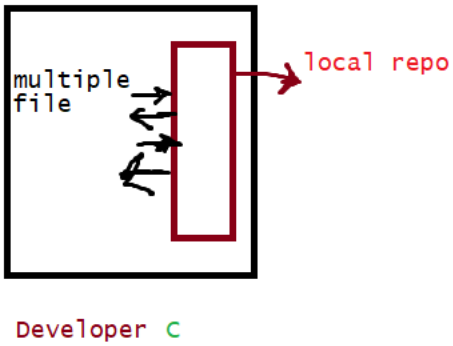
```
/D/Git Project/git_stash_deleting (master)
$ git stash clear
```

//Now there will be no stash.

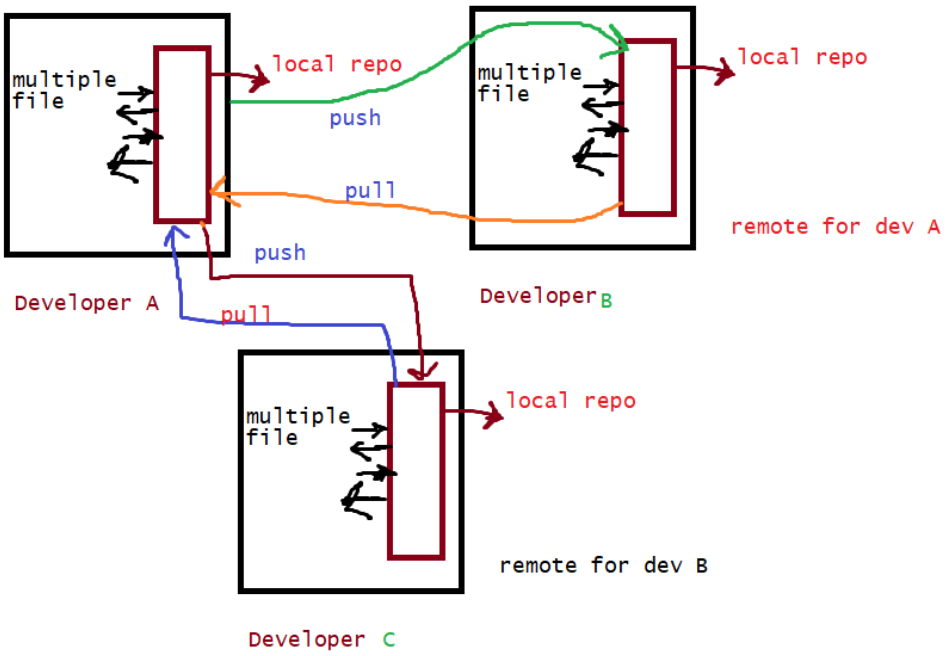
```
/D/Git Project/git_stash_deleting (master)
$ git stash list                                    #nothing..
```

Chap11:Need of remote repository?.

Till now we have created a lot of files and performed a lot of changes in our local repository (it created when we use ***git init*** command and local repository name is ***.git***).



Remote repository comes in picture when we have multiple developer and I want to share my code with them OR get code from Team member.



(Distributed means – Direct communication)

We know git is distributed version control system.

Note: w.r.t Developer A B's local repository is remote repository for him Vice versa for all other developer 😊. (Remote repo means which is not available on my machine).

As shown in second Image every developer can communicate with every other developer (same as A). It is called distributed design.

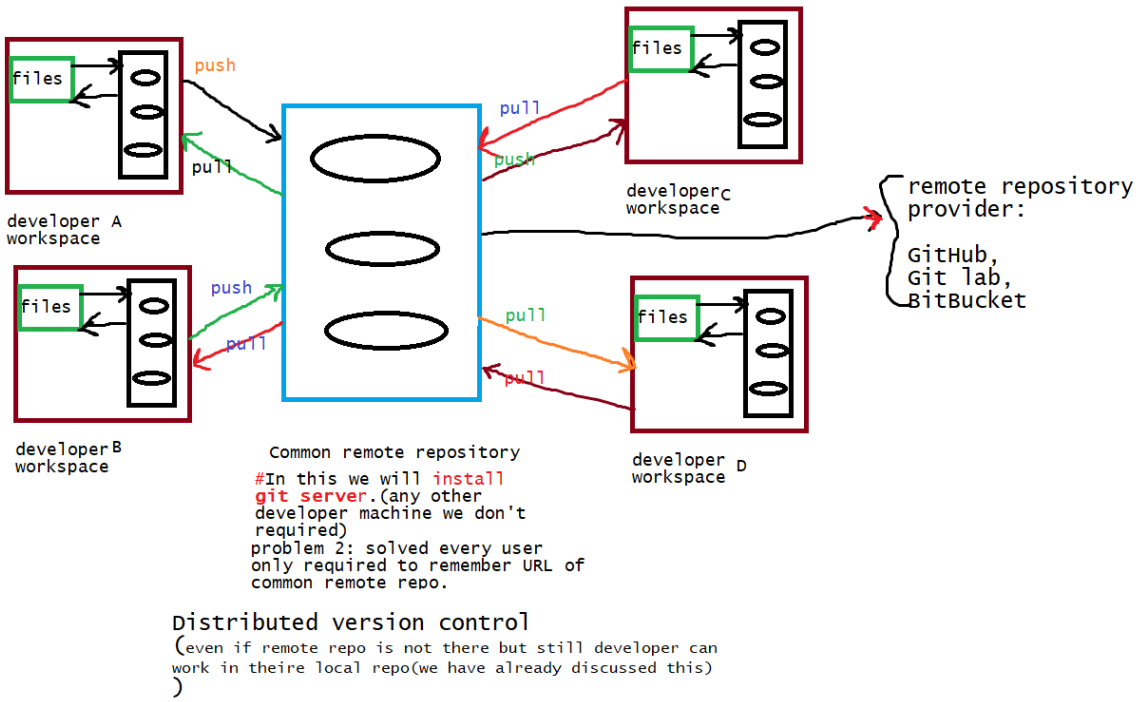
PUSH – sending code to remote
PULL – getting code from remote repo.

Above model is of distributed Type and as shown every developer can directly connect with every other and there are few disadvantages of this.

Problem with Direct Communication:

1. on every developer's machine GIT server should be installed. (Then only other developer can share code to remaining people. Till now we have installed on git client)
2. Developer A wants to communicate with Developer B.
   A should aware hostname (IP Address) of B without which he can not connect.
   He should no on which port number is this git server is running on peers' developer machine.
   so every people should aware every other developer hostname and port number.
3. If port number/hostname of one developer changed then communication will not happen.

We can solve above problem by using common remote repository. 😊



Big Companies = they are using their own common repository due to security.
We may need to work with multiple remote repositories at the same time. (ex. working on 3 project and all 3 clients are using different remote repository 😊).

Chap12:Git hub account and remote repository creation , git remote and push command.

Topic:

1. How to create FREE account in GitHub?
2. How to create remote repository in GitHub?
3. How to work with this remote repository?

Commands:

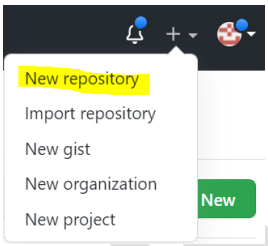
git remote
git push
git clone
git pull
git fetch

A) How to create FREE account in GitHub?

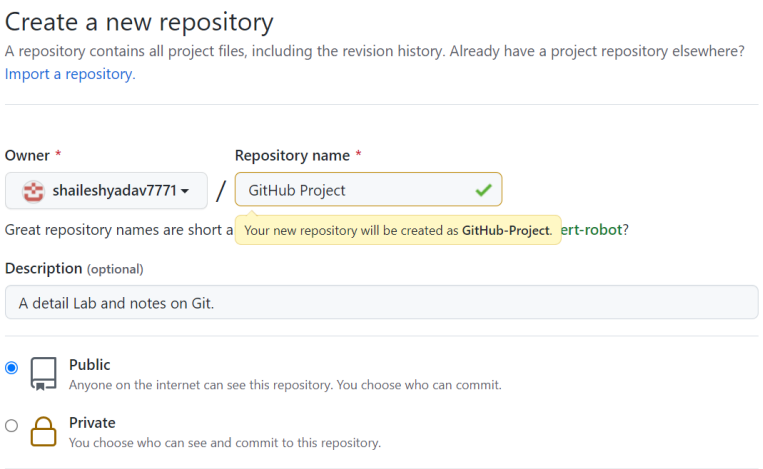
- Self-explanatory 😊(skipping it).
  visit https://github.com/ > register.

B) How to create remote repository in GitHub?

Click on + symbol

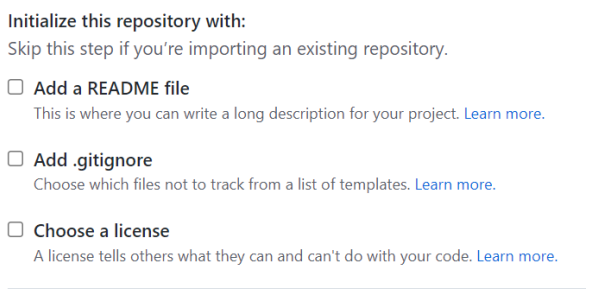


//Note: import repository means if we already have remote repository in Git lab, Bitbucket are any other platform. we don't have so creating new.

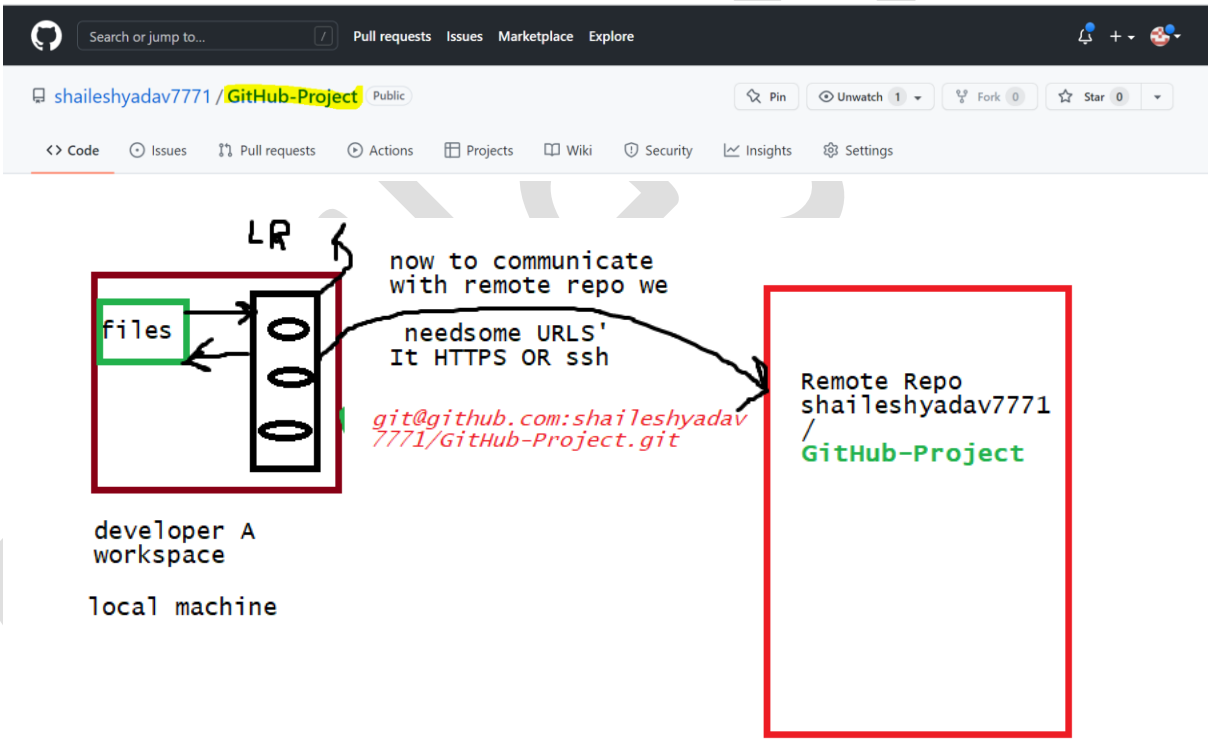


Repository name is our project folder name, so we need to create meaning full name. Public repository means everyone can see the data but only few members can commit whom write access given.

Private repository: Read and write both are performed by people who have respective access. (More security)- (It is paid service).



Note: We already have local repository in our machine so ignoring initialization part.
Alright now my remote repository is ready



C) How to work with this remote repository:

a) git remote:

Note: first if we want to connect with local repository then we need to configure remote repository with our local repository (sync) then we can work on it.
To configure remote repository to our local repository we will use git remote command.

Command:

```
$ git remote add remote_repository_url
```

Note: In above command we have not used ***alias name*** so going further for everything we need to use ***remote_repository_url*** a lengthy name 😊. So, we can use some alias name for this.
So, our command become:

```
$ git remote add origin remote_repository_url
```

So in our case,

```
$ git remote add origin git@github.com:shaileshyadav7771/GitHub-Project.git
```

Note: It is not compulsory to use origin so based on our need we can use any name, but general convention is to use origin.

How to check git remote?
If we use **`$ git remote`** `without any option/argument`, then it will show all the defined alias name.
To see URL's also we need to use:
`$ git remote -v`
//It will show alias name along with URL's name.

---

b) git push:

git push command can be used to push our code from local repository to local repository.

```
$ git push remote_repository_name <branch>

Example:
$ git push origin master
//instead of master we can have some other local repository.
```

Practical:
Now our remote repository is ready. Here I will push the exercise code what we discussed as a part of this course.

```
$ pwd
/D/Git Project

/D/Git Project (master)
$ git init
Reinitialized existing Git repository in D:/Git Project/.git/
```

//there was a lot of files in different folder was in untracked status and few were not committed so I have committed all and now working tree is clean.

```
/D/Git Project (master)
$ git log --oneline
37ea7a7 (HEAD -> master) final changes
2e9bba4 final saving
64db510 final saving
1f43960 saving all
a93e4a6 Adding all the data.
```

Step1: configure remote repository so first let me check if there is any remote repository are there OR not?

```
/D/Git Project (master)
$ git remote
```

So not associated so let me add.

```
/D/Git Project (master)
$ git remote add origin git@github.com:shaileshyadav7771/GitHub-Project.git
```

Now to verify:

```
$ git remote
origin
```

//now to check remote repository URL.

```
/D/Git Project (master)
$ git remote -v
origin  git@github.com:shaileshyadav7771/GitHub-Project.git (fetch)
origin  git@github.com:shaileshyadav7771/GitHub-Project.git (push)
```

step2: Now I want to send my code from local repository to remote repository.

```
/D/Git Project (master)
$ git branch
* master

Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.
```

Solution: Please check - https://docs.github.com/en/authentication/connecting-to-github-with-ssh/checking-for-existing-ssh-keys

Alright so I have generated ssh key and added it to my github account.

```
/D/Git Project (master)
$ git push origin master
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 8 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (19/19), 3.11 KiB | 3.11 MiB/s, done.
Total 19 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), done.
To github.com:shaileshyadav7771/GitHub-Project.git
 * [new branch]      master -> master
```

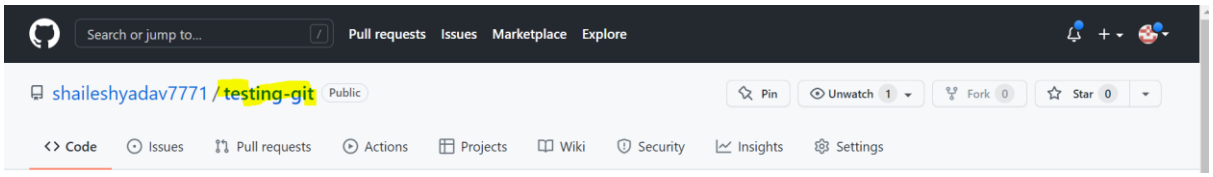
So, in above Enumerating objects means we are sending 19 objects.

Now just refresh and our project will be available there.
So now if anyone required, they could see and get the code from GitHub.

In Public repository: Total repo size = 2GB Max allowed file size: 100MB

How to work with multiple remote repository?

Let me create another remote repository:
//now we are having only one remote repository.




```
/D/Git Project (master)
$ git remote
origin

/D/Git Project (master)
$ git remote add origin git@github.com:shaileshyadav7771/testing-git.git
error: remote origin already exists.
```

So, we can see that origin alias name is already used so let us different name.

```
/D/Git Project (master)
$ git remote add shailesh_remote git@github.com:shaileshyadav7771/testing-git.git

/D/Git Project (master)
$ git remote
origin
shailesh_remote
```

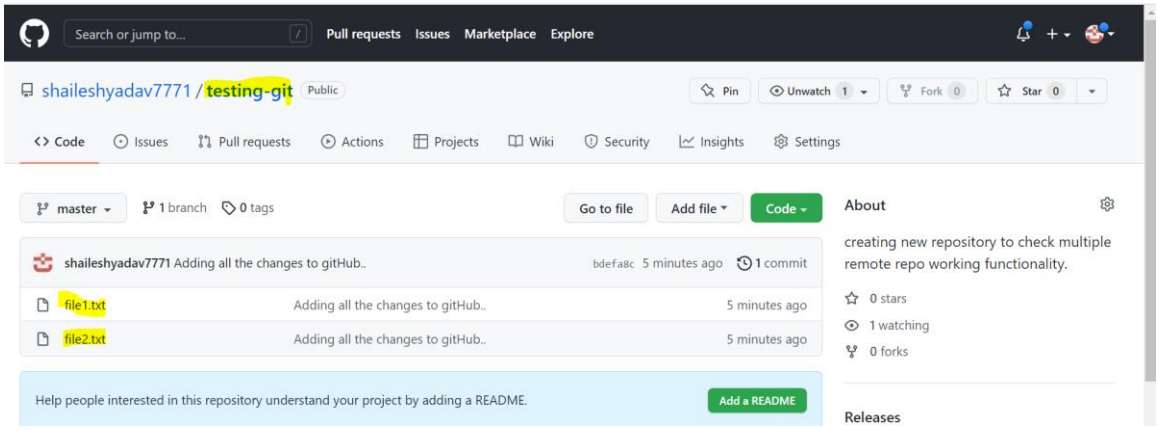
//Now we can see that a new alias created for our "*Shailesh_remote*".

To see the details.

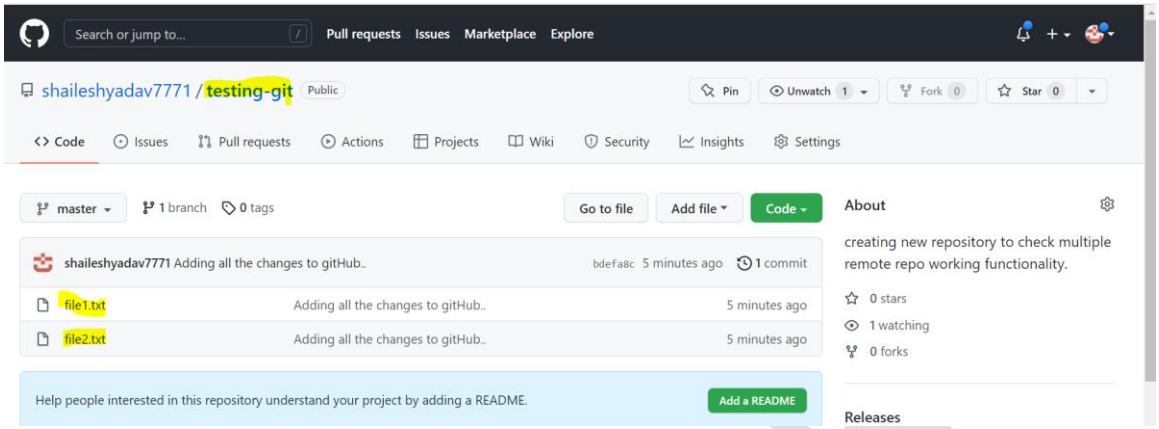
```
/D/Git Project (master)
$ git remote -v
origin  git@github.com:shaileshyadav7771/GitHub-Project.git (fetch)
origin  git@github.com:shaileshyadav7771/GitHub-Project.git (push)
shailesh_remote git@github.com:shaileshyadav7771/testing-git.git (fetch)
shailesh_remote git@github.com:shaileshyadav7771/testing-git.git (push)
```

//I have created a new project and I'll push into new remote repo.

```
$ git push shailesh_remote master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 318 bytes | 318.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:shaileshyadav7771/testing-git.git
 * [new branch]      master -> master
```

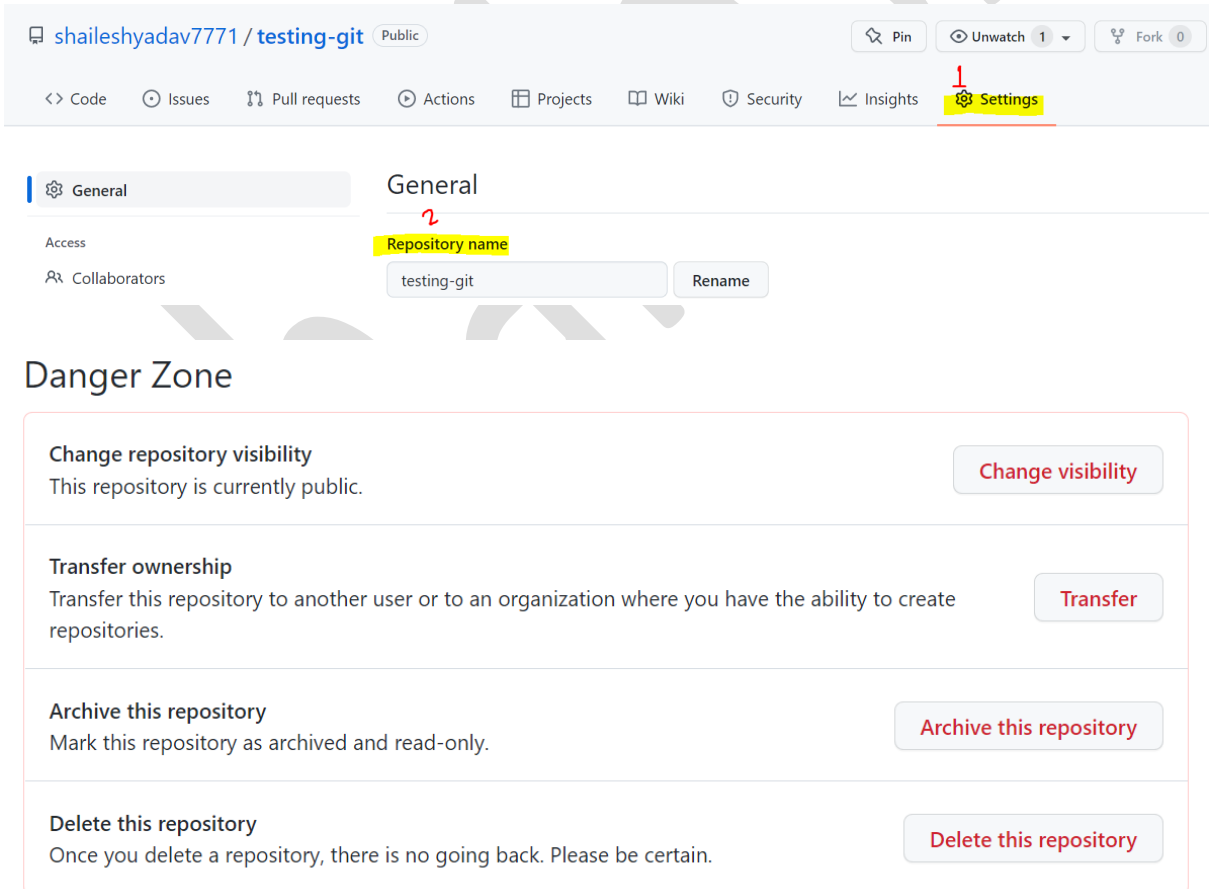


Chap12:git clone command.



Now in this chap we will discuss how to get/clone the remote repository how to rename it and also how to make public repository to private repository 😊.

For rename click on settings >



So, from change repository visibility we can change it from public to private. And same way we can change ownership of it.

c) git clone:

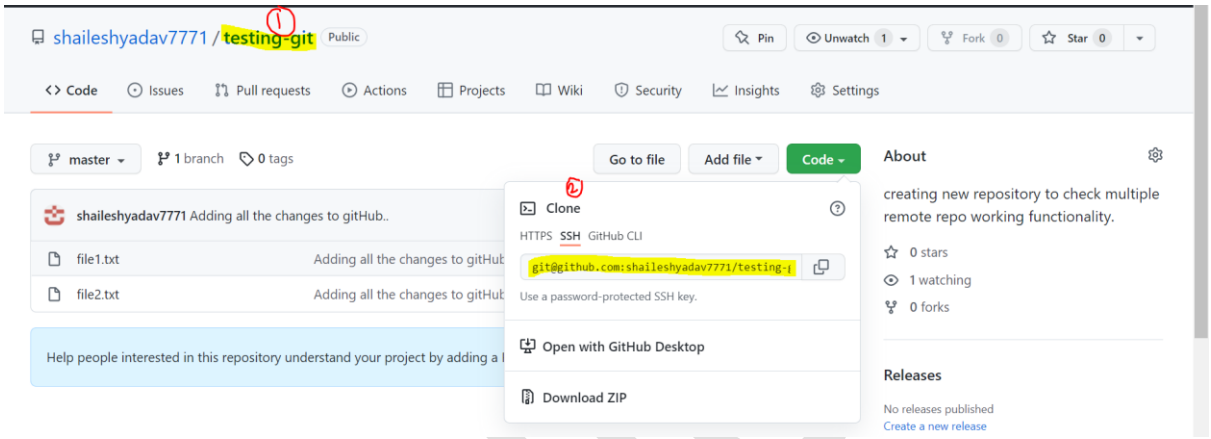
we can use git clone command to clone complete project from remote repository to our local repository.

Suppose as shown below some project has been already developed and it's remote repository is available and I am new Joiner then no problem simply I will clone all the project data to my local repository. 😊

```
$ git clone <remote-repository-URL>
```

Git clone will Create new local repository with all files and history of remote repository 😊.

Practical:
Now as shown below I am already having one remote repository so we will clone it.




```
/D
$ mkdir new_developer
```

//now I am new developer so let me create new folder and then clone above project.

```
/D
$ git clone git@github.com:shaileshyadav7771/testing-git.git
Cloning into 'testing-git'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

Now we can see all files.

```
/D/testing-git (master)
$ ls
file1.txt  file2.txt
```

```
/D/testing-git (master)
$ git log --oneline
bdefa8c (HEAD -> master, origin/master, origin/HEAD) Adding all the changes to
gitHub..
```

Note: We are not required to create local repository 😊 let me show It will come automatically.

```
/D/testing-git (master)
$ ls -la
total 10
drwxr-xr-x 1 GMX+ 4096  0 Feb 26 17:59 ./
drwxr-xr-x 1 GMX+ 4096  0 Feb 26 17:59 ../
drwxr-xr-x 1 GMX+ 4096  0 Feb 26 17:59 .git/
-rw-r--r-- 1 GMX+ 4096 25 Feb 26 17:59 file1.txt
-rw-r--r-- 1 GMX+ 4096 25 Feb 26 17:59 file2.txt
```

So, we can see. git (local repository is already present).

```
/D/testing-git (master)
$ cat file1.txt file2.txt
first line in file1.txt
first line in file2.txt
```

Note: here we got project folder name same as the remote repository name. So, It is possible to change it?
Yes.
We need to use below command.

```
$ git clone URL'S <directory-name-which we want to set>
```

Example:

```
/D/new_developer
$ git clone git@github.com:shaileshyadav7771/testing-git.git shailesh-project
Cloning into 'shailesh-project'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.
```

```
/D/new_developer
$ ls
shailesh-project/

/D/new_developer/shailesh-project (master)
$ ls
file1.txt  file2.txt
```

//so, we can see that now our folder name got changed and It is ***shailesh-project***.

Que: Before using git clone command, is it required to use git init command or not?
Ans: No, because git clone itself is providing it. 😊.

Que2: In how many ways we can create local repository?
Ans: There are two ways.

1. Empty local repository by using ***git init*** command.
2. Non-empty local repository by using ***git clone*** command.

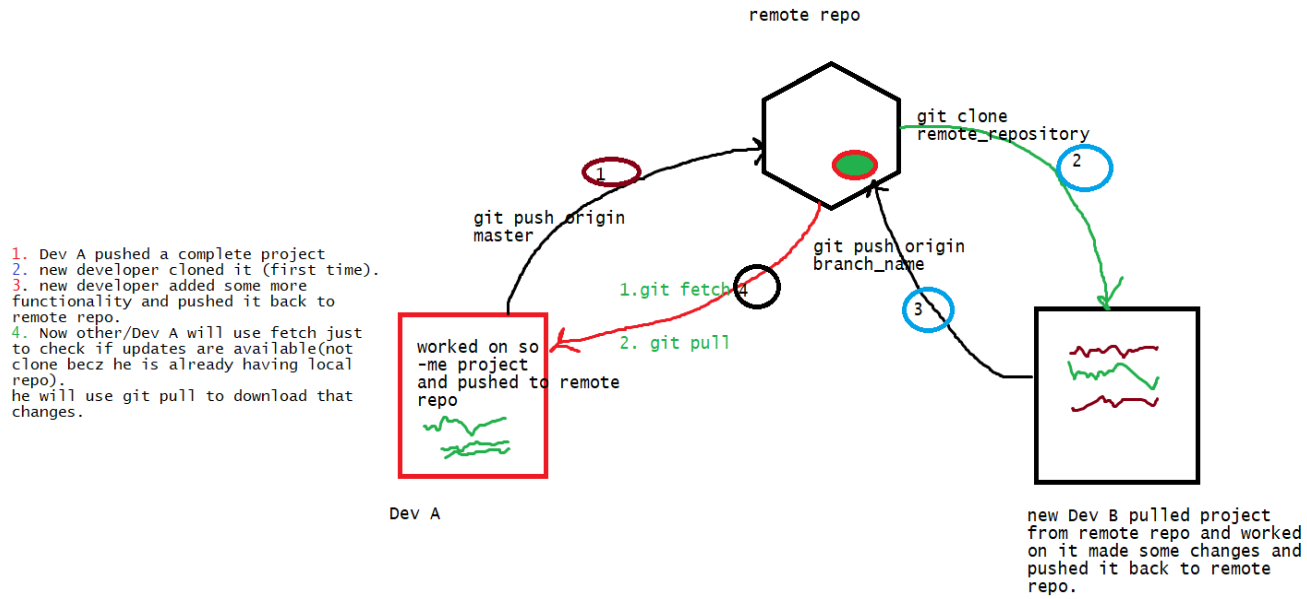
Chap13:git fetch and pull command.

Topic covered:
git remote = > To configure remote repository to the local repository
We can also get remote repository information.
git push = > To make changes/updates available from local repo to remote repo.
git clone = > To clone remote repository into our local repository.

Suppose as shown in below diagram Developer A worked on some project and in order to share code with it's Team it pushed it to remote repository. Now new member will clone it and assume he made some changes in it and committed/push back those changes back to remote repository.

Note: Now Developer A will not have that latest update and for that he is having 2 options 😊.

`$ git fetch`	`$ git pull`
//To check whether updates are available or not. It is not compulsory command just to check if any updates are available or not (we can directly use pull command 😊).	// To get updates from remote repository into local repository.



Que: Diff between git clone and git pull?
Ans: for first time (I am new member so) : ***$ git clone***
For already existing member: ***$ git pull***

We can say,
git pull = fetch + merge

Practical:

```
/D
$ mkdir dev_A_workspace

/D
$ cd dev_A_workspace/
```

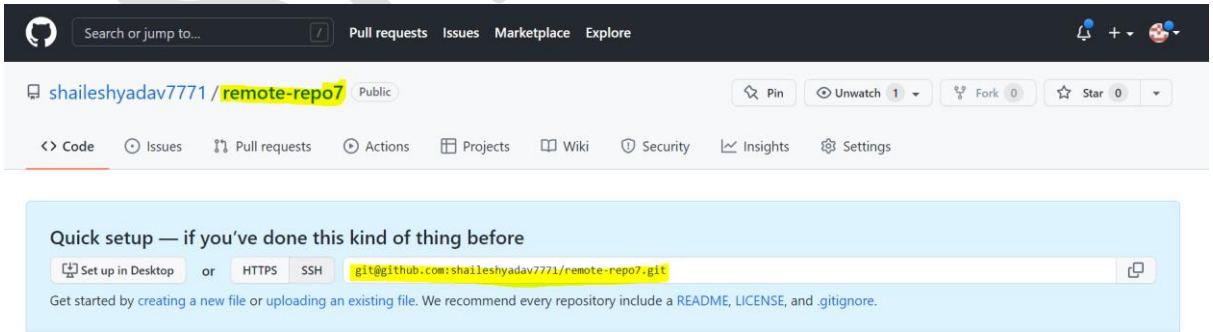
//I need to create empty repository in order to start a project so

```
/D/dev_A_workspace
$ git init
Initialized empty Git repository in D:/dev_A_workspace/.git/

/D/dev_A_workspace (master)
$ echo "first line in file1 by Dev A" > file1.txt

/D/dev_A_workspace (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        file1.txt

/D/dev_A_workspace (master)
$ git add .;git commit -m "By Dev-A"
[master (root-commit) 793f0c5] By Dev-A
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
```

So how many files' Dev A created and how many commits he did?
Ans: 1 file & 1 commit:

```
/D/dev_A_workspace (master)
$ ls
file1.txt

/D/dev_A_workspace (master)
$ git log --oneline
793f0c5 (HEAD -> master) By Dev-A
```

Note: Now he will push this code to remote repository so here let us create new remote repo for this.



So created a new remote repository with remote-repo7 name.

So, before Dev A push code first he needs to register the remote repo in his local repository.

```
/D/dev_A_workspace (master)
$ git remote
```

//now there is no remote configured.

```
/D/dev_A_workspace (master)
$ git remote add origin git@github.com:shaileshyadav7771/remote-repo7.git
```
//here origin is alias name 😊.

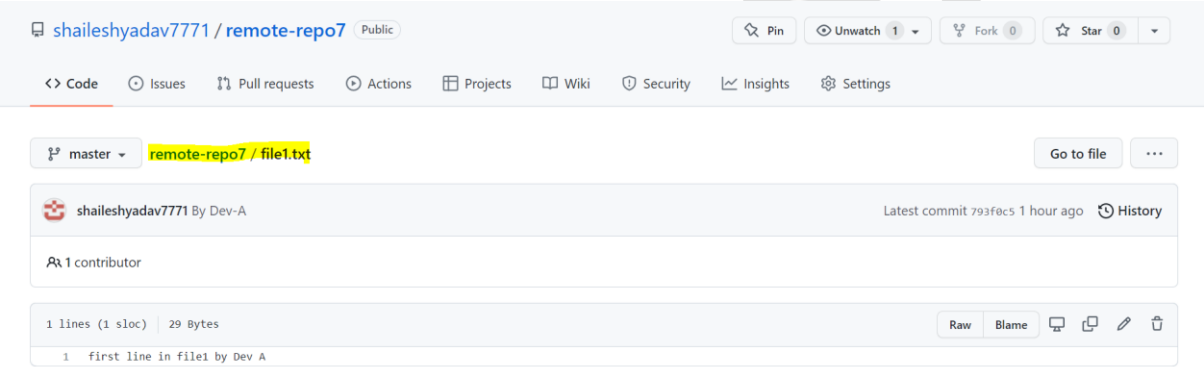
Now let us check

```
/D/dev_A_workspace (master)
$ git remote -v
origin  git@github.com:shaileshyadav7771/remote-repo7.git (fetch)
origin  git@github.com:shaileshyadav7771/remote-repo7.git (push)
```

so here we can see that origin is pointing to our remote repository.

Now let me push it to remote repo.

```
/D/dev_A_workspace (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 243 bytes | 243.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:shaileshyadav7771/remote-repo7.git
 * [new branch]      master -> master
```



Alright now we can see the file1.txt is present in remote repository. (All changes done by developer A is came to remote-repo7).

Now **Dev B** is new person and he want to work on this project 😊.

```
/D
$ mkdir dev_B_work_space

/D
$ cd dev_B_work_space/
```

//git init is not required (Dev B is not required to create he will get it from remote repository).

Note: Sir I am cloning so do I need to configure remote repository 😊 No Sir we are using git clone URL and in URL itself we are telling. (For push it is required but for the get not needed).

For example, if we use we will get:

```
/D/dev_B_work_space
$ git remote
fatal: not a git repository (or any of the parent directories): .git
```

```
/D/dev_B_work_space
$ git clone git@github.com:shaileshyadav7771/remote-repo7.git
Cloning into 'remote-repo7'...
```

```
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

Let us cross check:

```
$ ls
remote-repo7/

/D/dev_B_work_space
$ cd remote-repo7/

/D/dev_B_work_space/remote-repo7 (master)
$ ls
file1.txt

/D/dev_B_work_space/remote-repo7 (master)
$ cat file1.txt
first line in file1 by Dev A

/D/dev_B_work_space/remote-repo7 (master)
$ git log --oneline
793f0c5 (HEAD -> master, origin/master, origin/HEAD) By Dev-A
```

Now Dev-B will work, and he will make changes in the file1.txt.

```
/D/dev_B_work_space/remote-repo7 (master)
$ echo "This line added by Dev-B " >> file1.txt
```

Let us add and commit it

```
/D/dev_B_work_space/remote-repo7 (master)
$ git add .;git commit -m "commit on file1 by B"
[master 96ec477] commit on file1 by B
 1 file changed, 1 insertion(+)

/D/dev_B_work_space/remote-repo7 (master)
$ echo "new file-enhancement Added by Dev B" > file2.txt

/D/dev_B_work_space/remote-repo7 (master)
$ git add . ; git commit -m "commit by B -file2.txt"
warning: LF will be replaced by CRLF in file2.txt.
The file will have its original line endings in your working directory
[master 6b1598c] commit by B -file2.txt
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt
```

Changes by Dev A:

1. Added extra line in file1.txt
2. He created a new file2.txt and did 2 commits.

Note: All above updates are available to **Dev B** machine and not visible to developer A.
As shown in Dev-A machine we are having only 1 file.

```
/D/dev_A_workspace (master)
$ ls
file1.txt
```

but dev-A is having two files (change sin file1.txt and file2.txt) 😊.

```
/D/dev_B_work_space/remote-repo7 (master)
$ ls
file1.txt  file2.txt
```

Now we want these changes (done by DEV-B) to be there in the remote repository so that others/Dev-A can get it.
Note: We are working on Dev-B machine who cloned the remote repo so he didn't required to configure remote branch (origin will be automatically configured for fetch and push) so Instead of ***$ git push origin master*** we can use ***$ git push***.

for some other testing purpose, I am creating one new branch in Dev A machine.

```
/D/dev_A_workspace (master)
$ git branch test123
```

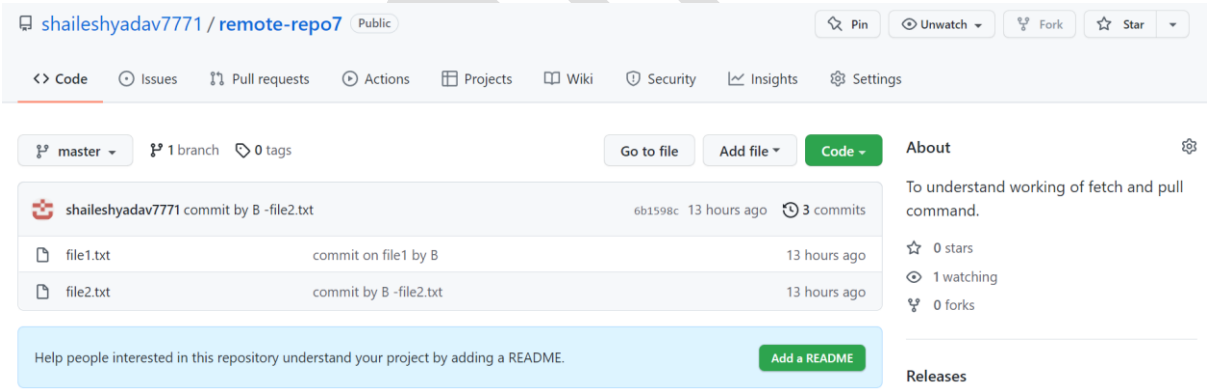
//now in branch test123 We will have all the files from Dev-A master branch.

```
/D/dev_A_workspace (test123)
$ ls
file1.txt
```

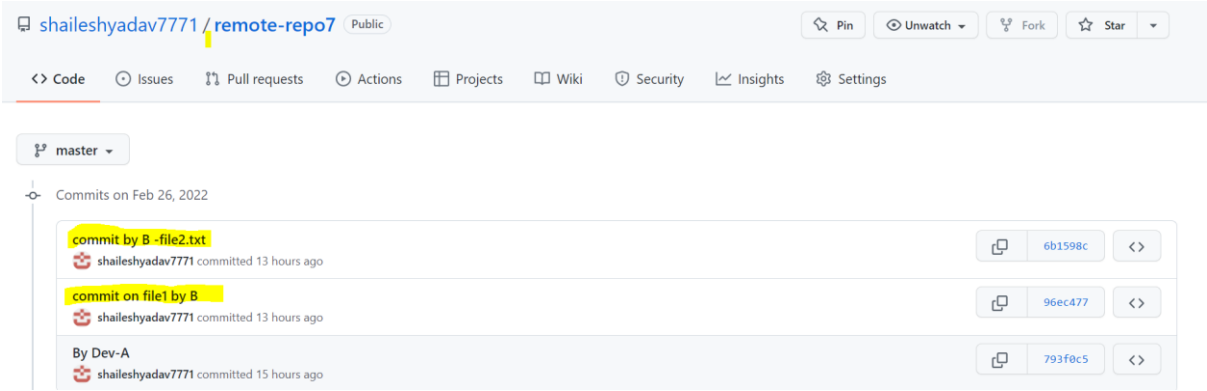
Now let us push changes from Dev-B machine.

```
/D/dev_B_work_space/remote-repo7 (master)
$ git push origin master
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (6/6), 580 bytes | 290.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:shaileshyadav7771/remote-repo7.git
   793f0c5..6b1598c  master -> master
```

Alright now let us check whether changes are available OR not in remote repository?



So now we can see below there are 3 commits (2nd and 3rd commit by Dev-B).



Now **Dev-A** want to get above changes from remote repo by using git fetch and git pull (if he is new member then need to clone it 😊).

Now first let me check if there are any objects/files updates available in remote repo (from Dev-A) from master branch.

```
/D/dev_A_workspace (master)
$ git fetch origin
//It will show o/p if update available.
```

Now let us update it (I want remote repo data and want to merge it).

```
/D/dev_A_workspace (master)
$ git pull origin master
From github.com:shaileshyadav7771/remote-repo7
 * branch            master     -> FETCH_HEAD
Updating 793f0c5..6b1598c
Fast-forward
 file1.txt | 1 +
 file2.txt | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 file2.txt
```

so we can see that 2 insertions happen one is modification in file1.txt and second is addition of file2.txt. 😊.

Now if we check we will get 2 files.

```
/D/dev_A_workspace (master)
$ ls
file1.txt  file2.txt
```

```
/D/dev_A_workspace (master)
$ cat file1.txt
first line in file1 by Dev A
This line added by Dev-B

/D/dev_A_workspace (master)
$ cat file2.txt
new file-enhancement Added by Dev B
```

we will get complete commit history:

```
/D/dev_A_workspace (master)
$ git log --oneline
6b1598c (HEAD -> master, origin/master) commit by B -file2.txt
96ec477 commit on file1 by B
793f0c5 (test123) By Dev-A
```

FOR testing purpose, we have created a new branch now let us switch to it and see content 😊.

```
/D/dev_A_workspace (master)
$ git checkout test123
Switched to branch 'test123'
```

//so here we don't have latest changes so let us merge it with master branch.
Before that let me add some data in file1.txt

```
/D/dev_A_workspace (test123)
$ vi file1.txt

/D/dev_A_workspace (test123)
```

```
$ cat file1.txt
first line in file1 by Dev A
Adding this line from DEV-A machine and through test123 branch.

/D/dev_A_workspace (test123)
$ git add .;git commit -m "Adding this line from DEV-A machine and through
test123 branch"
[test123 b3ca6dc] Adding this line from DEV-A machine and through test123 branch
 1 file changed, 1 insertion(+)

/D/dev_A_workspace (test123)
$ git pull origin master
From github.com:shaileshyadav7771/remote-repo7
 * branch            master     -> FETCH_HEAD
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Now we can see that there is conflict okay so let us resolve it.
Considering both the changes. 😊

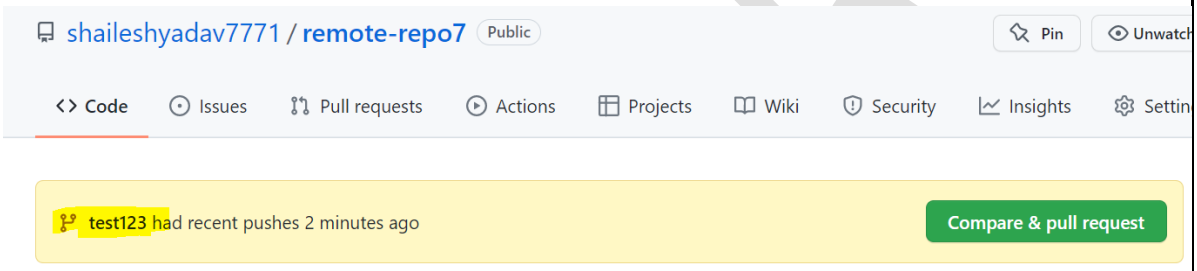
Now once done then we need add files and commit it.

```
/D/dev_A_workspace (test123|MERGING)
$ git add .; git commit -m "fetched from master and working from DE-A test123
branch and considering both the changes."
[test123 242c085] fetched from master and working from DE-A test123 branch and
considering both the changes.


/D/dev_A_workspace (test123)
$ git log --oneline
242c085 (HEAD -> test123) fetched from master and working from DE-A test123
branch and considering both the changes.
b3ca6dc Adding this line from DEV-A machine and through test123 branch
6b1598c (origin/master, master) commit by B -file2.txt
96ec477 commit on file1 by B
793f0c5 (test1234) By Dev-A
```

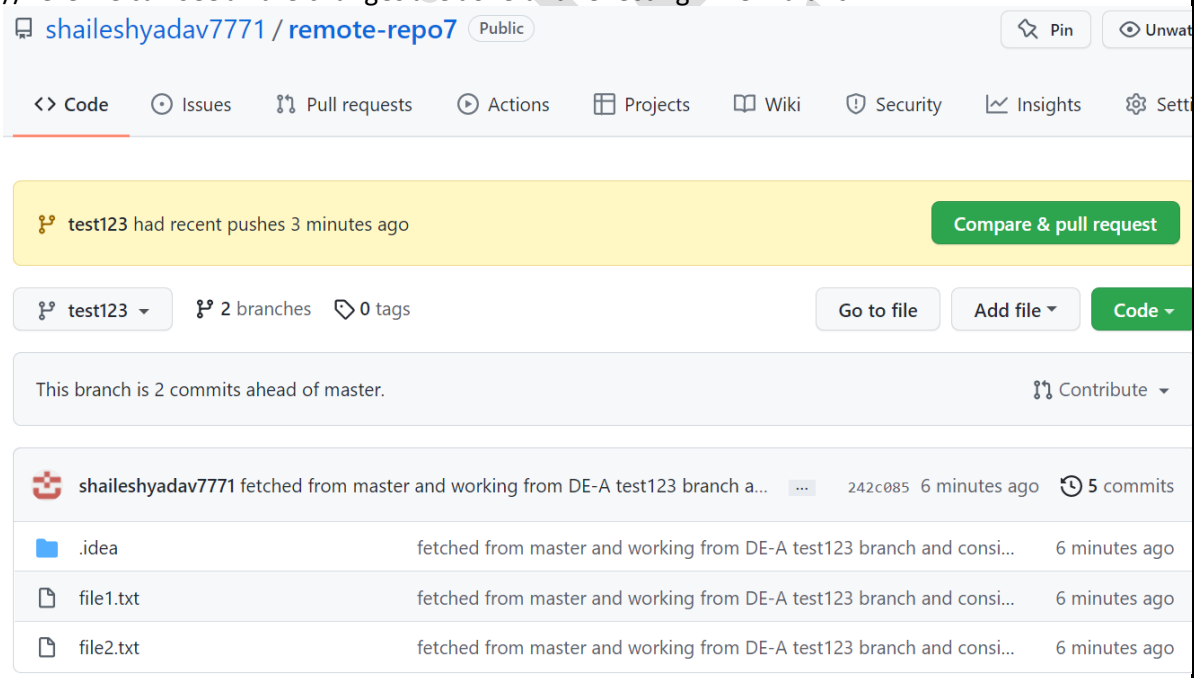
//Now let us push these changes to remote repo.

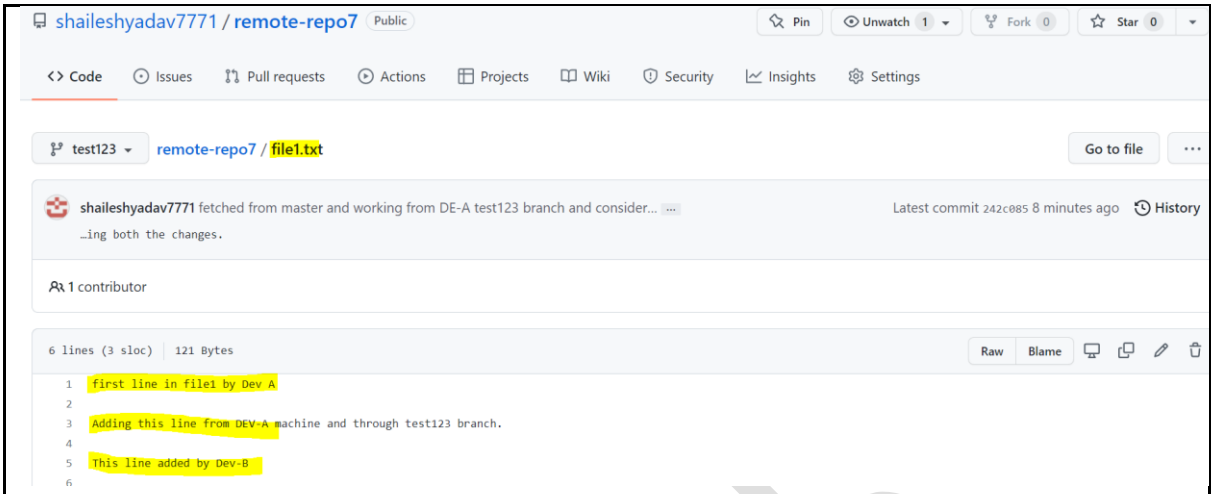
```
/D/dev_A_workspace (test123)
$ git push origin test123
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (15/15), 1.89 KiB | 1.89 MiB/s, done.
Total 15 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'test123' on GitHub by visiting:
remote:      https://github.com/shaileshyadav7771/remote-repo7/pull/new/test123
remote:
To github.com:shaileshyadav7771/remote-repo7.git
 * [new branch]      test123 -> test123
```

Note: This time we are pushing changes from test123 branch 😊.

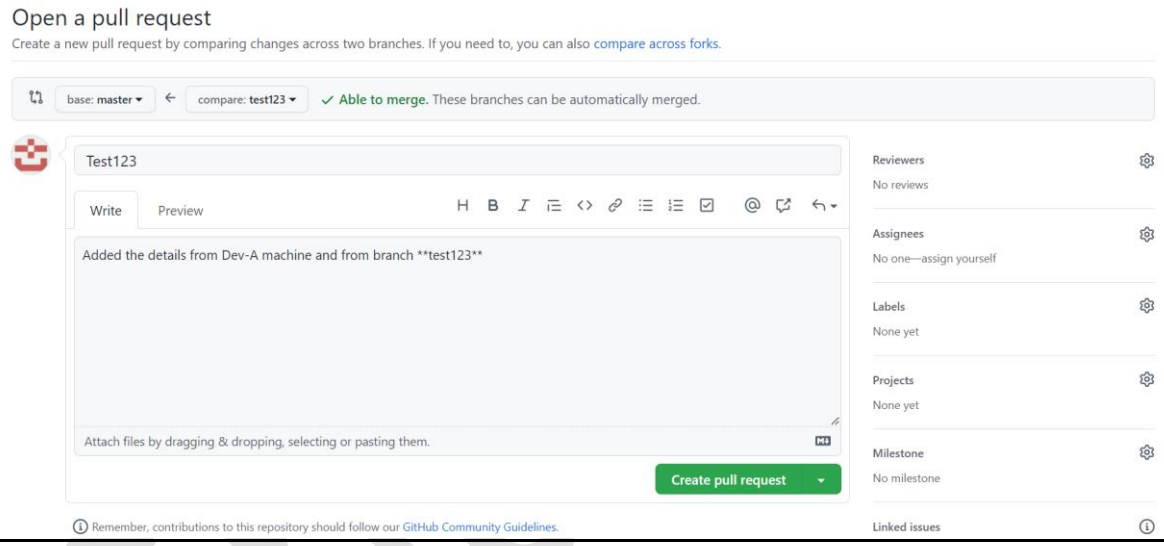


//here we can see all the changes are done and reflecting in new branch

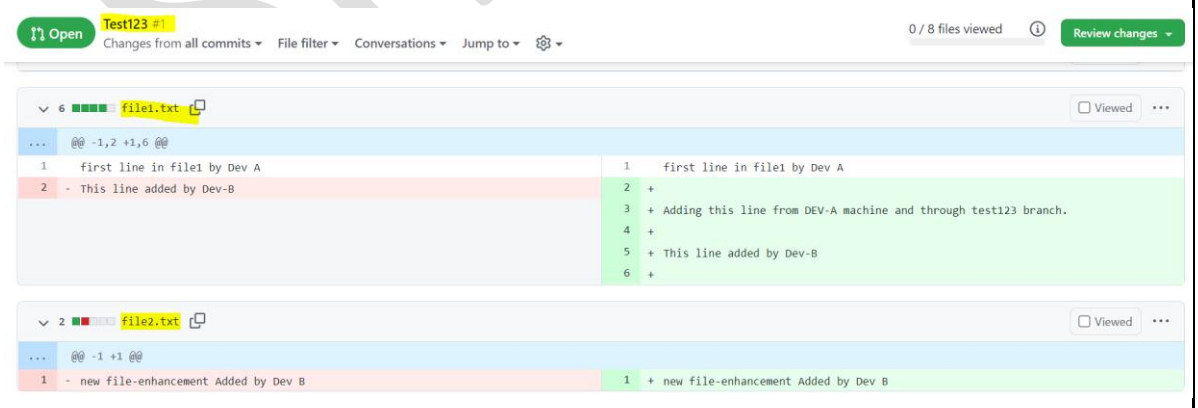


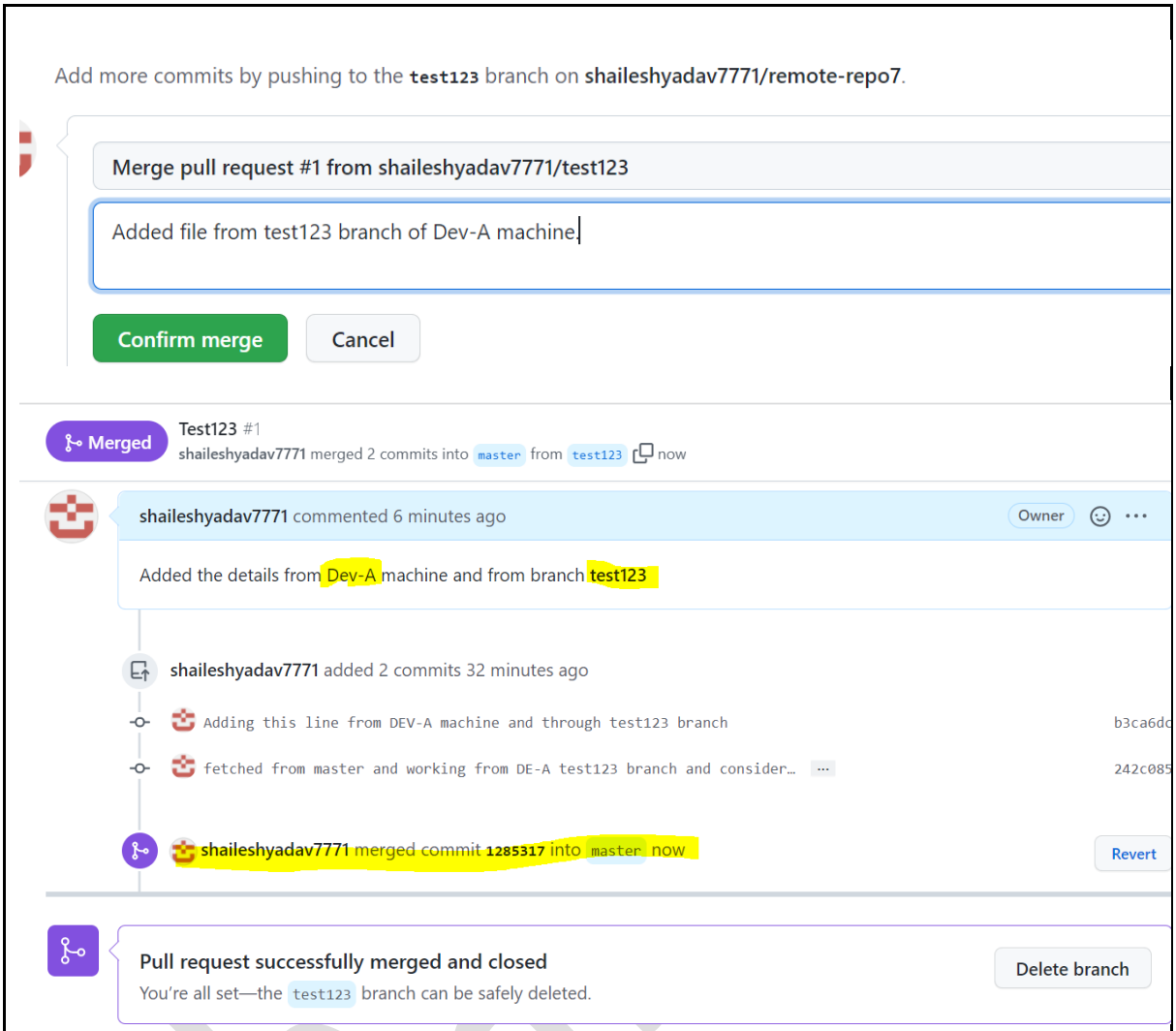


Now let us create a pull request.



After clicking on pull request.





Now the changes are merged to master branch so if we want, we can delete it.

Now let check our master branch (which is still having old data because we have created a new branch and updated that branched and pushed data to origin master branch 😊).

```
/D/dev_A_workspace (test123)
$ git checkout master
Switched to branch 'master'

/D/dev_A_workspace (master)
$ ls
file1.txt  file2.txt

/D/dev_A_workspace (master)
$ cat file1.txt
first line in file1 by Dev A
This line added by Dev-B

/D/dev_A_workspace (master)
$ cat file2.txt
new file-enhancement Added by Dev B
```

now let us first check if any updates are available from origin.

```
/D/dev_A_workspace (master)
```

```
$ git fetch origin
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 652 bytes | 326.00 KiB/s, done.
From github.com:shaileshyadav7771/remote-repo7
   6b1598c..1285317  master     -> origin/master
```

Okay Its s available so let us use pull command (fetch + merge).

```
/D/dev_A_workspace (master)
$ git pull origin master
From github.com:shaileshyadav7771/remote-repo7
 * branch            master     -> FETCH_HEAD
Updating 6b1598c..1285317
Fast-forward
 .idea/.gitignore                                  | 3 +++
 .idea/dev_A_workspace.iml                         | 8 ++++++++
 .idea/inspectionProfiles/profiles_settings.xml | 6 ++++++
 .idea/misc.xml                                    | 4 ++++
 .idea/modules.xml                                 | 8 ++++++++
 .idea/vcs.xml                                    | 6 ++++++
 file1.txt                                         | 6 +++++-
 file2.txt                                         | 2 +-
 8 files changed, 41 insertions(+), 2 deletions(-)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/dev_A_workspace.iml
 create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/modules.xml
 create mode 100644 .idea/vcs.xml

/D/dev_A_workspace (master)
$ ls
file1.txt  file2.txt
```

now all the files are showing in Dev-A branch.

```
/D/dev_A_workspace (master)
$ cat file1.txt
first line in file1 by Dev A

Adding this line from DEV-A machine and through test123 branch.

This line added by Dev-B
```

//now initially we have also created one branch with test1234 so I have updated master branch now I will merge test1234 branch with master.

```
/D/dev_A_workspace (master)
$ git checkout test1234
Switched to branch 'test1234'

/D/dev_A_workspace (test1234)
$ ls
file1.txt

/D/dev_A_workspace (test1234)
$ cat file1.txt
first line in file1 by Dev A
```

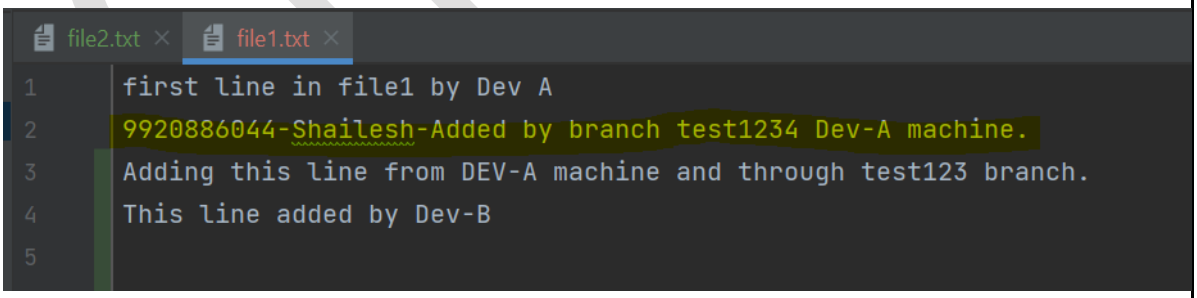
//now let us suppose Dev-A has worked on file1.txt and already made some changes.

```
/D/dev_A_workspace (test1234)
$ cat file1.txt
first line in file1 by Dev A
9920886044-Shailesh-Added by branch test1234 Dev-A machine.
```

```
/D/dev_A_workspace (test1234)
$ git add .;git commit -m "made some changes in file1.txt from test1234 branch -
Dev-A machine."
[test1234 ead62b8] made some changes in file1.txt from test1234 branch -Dev-A
machine.

/D/dev_A_workspace (test1234)
$ git log --oneline
ead62b8 (HEAD -> test1234) made some changes in file1.txt from test1234 branch -
Dev-A machine.
793f0c5 By Dev-A


//now let us merge it with Dev-A test1234 branch.
```
(Note: my master branch is already updated.)

```
/D/dev_A_workspace (test1234)
$ git merge master
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

//now let me resolve conflict.

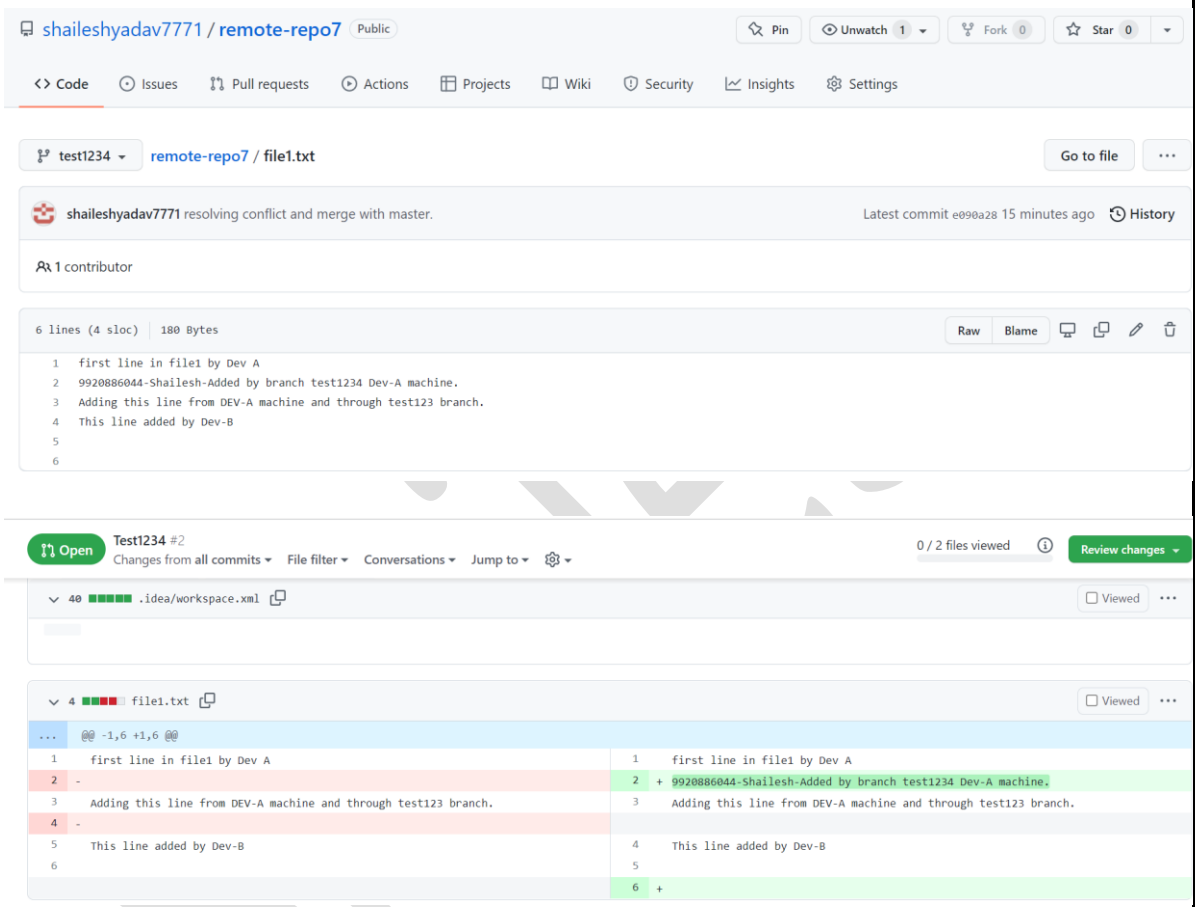


Now made changes so let us add and commit it. 😊

```
/D/dev_A_workspace (test1234|MERGING)
$ git add .;git commit -m "resolving conflict and merge with master."
[test1234 e090a28] resolving conflict and merge with master.

/D/dev_A_workspace (test1234)
$ git push origin test1234
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 8 threads
Compressing objects: 100% (8/8), done.
```

```
Writing objects: 100% (9/9), 1.61 KiB | 1.61 MiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'test1234' on GitHub by visiting:
remote:      https://github.com/shaileshyadav7771/remote-repo7/pull/new/test1234
remote:
To github.com:shaileshyadav7771/remote-repo7.git
 * [new branch]      test1234 -> test1234
```

Now let us check it on UI.



So, we can see the changes are reflecting so let me merge it with master (remote repo).

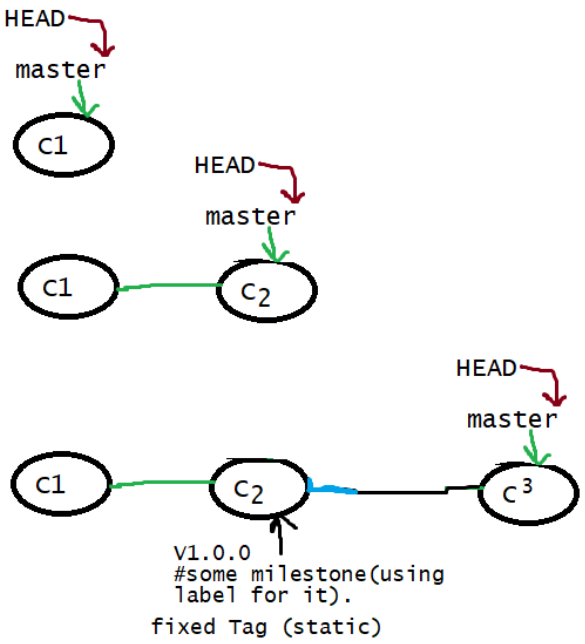
Note: To perform push operation local repository and remote repository should be in sync (if there will be any changes in remote repo then first, we need to do pull (conflict resolution) and then push). That is local repository should be up to date with remote repository.
Before calling push operation we must perform pull operation.

If any conflict is there, we need to resolve it (sometime may be required to check with person who committed 😊).

Chap14:git tagging -Light weight Tags.

Sir, real time example we are going for shopping there we see **tags** on clothes and other products which basically tell us about the product price 😊 and some more useful information.
In same way in repository, we are maintaining source code and commits.

Sir we know and as shown below when we create a commit let say c1 then master will be pointing to C1 (commit 1) and HEAD is symbolic reference pointing to master. When we create commit c2 then this master & HEAD will move to c2 commit and so on. (So, it is changing-not fixed).

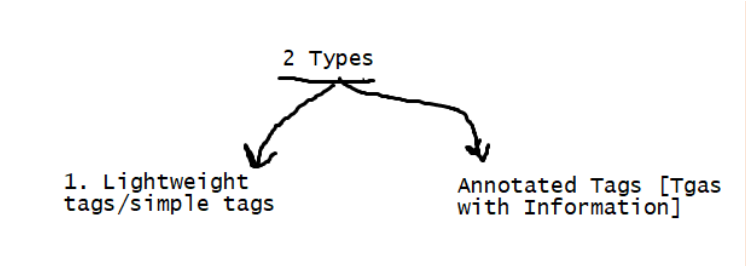


So as long as new commit created this HEAD and master will change (Dynamic references 😊).

Now sir suppose some Significant event/milestone in repository happen at C2 commit and I want to use some label for this to indicate it.

Tag is nothing but a label OR mark to particular commit in our repository.
Tag is a static and permanent reference to a particular commit.
In general, we can use tags concept for release versions. 😊

Types of Tags:



Generally, we use Annotated Tags in real time but let us discuss both in detail.

Lightweight tags/simple tags:

Syntax:

```
$ git tag <tag_name>
e.x: git tag V1.0.0
```

above we are not defining any commit ID so above tags will be assign to the latest commit.

//Now but sir I want to define it for some specified/old commit it is possible?
Yes,

How to list available Tags:
Ans: `$ git tag -l` **OR** `$ git tag -list`

Where tags will be stored and is it possible to delete tags?
Yes, we can do (like CRUD operation we do in database 😊).
.git/refs/tags
= > All our light weight tags will be stored.

For delete = > git tag -d <tag-name> **OR** git tag --delete <tag-name>

Note: Same is applicable for annotated tags but why we should go we will discuss 😊.

Practical:

```
/d
$ mkdir tagging

/d
$ cd tagging/
```

//to create empty local repo.

```
/d/tagging
$ git init
Initialized empty Git repository in D:/tagging/.git/
```

//now we need multiple commits.

```
/d/tagging (master)
$ echo "First Line" > file1.txt

/d/tagging (master)
$ git add .;git commit -m "first commit."
[master (root-commit) c162d32] first commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt

/d/tagging (master)
$ echo "First Line" > file2.txt

/d/tagging (master)
$ git add .;git commit -m "second commit."
[master 937f9ba] second commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt

/d/tagging (master)
```

```
/d/tagging (master)
$ echo "First Line" > file3.txt

/d/tagging (master)
$ git add .;git commit -m "3rd commit."
[master bce3b9d] 3rd commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt

/d/tagging (master)
$ echo "First Line" > file4.txt

/d/tagging (master)
$ git add .;git commit -m "4th commit."
warning: LF will be replaced by CRLF in file4.txt.
The file will have its original line endings in your working directory
[master 77ef69b] 4th commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt
```

so, we have total 4 commit and as discussed master & HEAD will point to forth commit.

```
/d/tagging (master)
$ git log --oneline
77ef69b (HEAD -> master) 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

So, tomorrow if new commit will come then HEAD & master (dynamic refs) will change and we cannot depend on it and want to change it with static tag.

```
/d/tagging (master)
$ git tag V-1.0.0
```

//As discuss above tag will be pointing to the latest commit which is commit 4.

Proof:

```
/d/tagging (master)
$ git log --oneline
77ef69b (HEAD -> master, tag: V-1.0.0) 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

So, we can see now the 4th commit is having tag V-1.0.0

Now let me create another commit.

```
/d/tagging (master)
$ echo "First Line" > file5.txt

/d/tagging (master)
$ git add .;git commit -m "5th commit."
[master 58189ba] 5th commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file5.txt

/d/tagging (master)
$ git log --oneline
58189ba (HEAD -> master) 5th commit.
77ef69b (tag: V-1.0.0) 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

So, we can see that master & HEAD moved but our defined tag is still pointing to 4th commit. 😊

List out total tags:

```
/d/tagging (master)
$ git tag --list
V-1.0.0
```

Where is it stored?

Ans: inside below files.

```
/d/tagging/.git/refs/tags (GIT_DIR!)
$ ls
V-1.0.0                                          #file

/d/tagging/.git/refs/tags (GIT_DIR!)
$ cat V-1.0.0
77ef69bdc9710udb0c1c3c387e4f3c7f558d1d4e #red colour is same commit check 4th
commit ID.
```

//Now let us create a tag for 5th commit ID.

```
/d/tagging (master)
$ git tag V1.1.0

/d/tagging (master)
$ git log --oneline
58189ba (HEAD -> master, tag: V1.1.0) 5th commit.
77ef69b (tag: V-1.0.0) 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

Note: for tagging we cannot use same name (*within repo tag name should be unique*).
Now we will have 2 tags. Example

```
/d/tagging (master)
$ git tag --list
V-1.0.0
V1.1.0
```

Deleting a Tag:

```
/d/tagging (master)
$ git tag --delete V1.1.0
Deleted tag 'V1.1.0' (was 58189ba)
```

So, this will delete tag, but commit will be there as it is 😊.

```
/d/tagging (master)
$ git log --oneline
58189ba (HEAD -> master) 5th commit.
77ef69b (tag: V-1.0.0) 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

Note: Semantic Versioning (for more details: *https://semver.org/*)

V-1.2.3

In above

1- Major version
2- minor version
3- patch Version to fix bugs 😊

Chap15:annotated Tags.

Sir in previous chapter we have discussed more about the lightweight/simple tags so here what is the need of discussing annotated tags and what are the disadvantages of simple tags we will see.

```
/d/tagging (master)
$ git tag -l
v-1.0.0
```

So let me remove above tag.

```
/d/tagging (master)
$ git tag -d v-1.0.0
Deleted tag 'v-1.0.0' (was 77ef69b)

/d/tagging (master)
$ git log --oneline
58189ba (HEAD -> master) 5th commit.
77ef69b 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

Now let me add one tag

```
/d/tagging (master)
$ git tag v-1.0.0

/d/tagging (master)
$ git log --oneline
58189ba (HEAD -> master, tag: v-1.0.0) 5th commit.
77ef69b 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

Alright now instead of using commit ID we can use our tag for more information.

```
$ git show  v-1.0.0
commit 58189bab9b92820fcba9e8215e3acaf390fe62e8 (HEAD -> master, tag: v-1.0.0)
Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>
Date:   Sun Feb 27 18:29:02 2022 +0530

    5th commit.

diff --git a/file5.txt b/file5.txt
new file mode 100644
index 0000000..603cb1b
--- /dev/null
+++ b/file5.txt
@@ -0,0 +1 @@
+First Line
```

Note: Tags is static reference to commit ID (so we can use it instead of commit ID).

Limitation of Light weight Tags/simple Tags:

-- - --- - - --- - - --- - - --- - - --- - --- -- - - - - - - -

1. It won't maintain any extra information like tagger name, date, description/message (It is just a level).

So, for above details we need to go for Annotated Tags 😊.

Annotated Tags:

-- - --- - - --- - - --- - - --- - - --- - --- -- - - - - - - -

1. It is exactly same as light weight Tag except that it holds information like.
a) tagger name, date, message, etc.

syntax:

```
$ git tag -a <tag-name> -m <tag-message>
```

#For annotated a tag object will be created to hold the value and in above command if we are not using <tag-message> then by default an editor will be open and ask for message 😊.

Now let me create one latest commit and for that we will create annotated tags.

```
/d/tagging (master)
$ echo "First Line" > file6.txt

/d/tagging (master)
$ git add .;git commit -m "6th commit."
[master b016355] 6th commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file6.txt
```

//create annotated Tags:

```
/d/tagging (master)
$ git tag -a V-1.1.0 -m "Release 1.1.0"
```

Now annotated Tag will be created for the latest commit.

```
/d/tagging (master)
$ git log --oneline
b016355 (HEAD -> master, tag: V-1.1.0) 6th commit.
58189ba (tag: V-1.0.0) 5th commit.
77ef69b 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

Now we know Annotated tag will provide some extra information.

Simple Tag	Annotated Tag

`$ git show V-1.0.0`
`commit 58189bab9b92820fcba9e8215e3acaf390fe62e8 (tag: V-1.0.0)`
`Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>`
`Date: Sun Feb 27 18:29:02 2022 +0530`
 	
`5th commit.`
 	
`diff --git a/file5.txt b/file5.txt`
`new file mode 100644`
`index 0000000..603cb1b`
`--- /dev/null`
`+++ b/file5.txt`
`@@ -0,0 +1 @@`
`+First Line`
 	
//info start directly from commit 😊.
	`$ git show V-1.1.0`
	`tag V-1.1.0`
	`Tagger: shaileshyadav7771 <shaileshyadav7958@gmail.com>`
	`Date: Sun Feb 27 20:29:42 2022 +0530`
	
	`Release 1.1.0`
	
	`commit b0163556e5591469dfa2788fa6f53449922867e3 (HEAD -> master, tag: V-1.1.0)`
	`Author: shaileshyadav7771 <shaileshyadav7958@gmail.com>`
	`Date: Sun Feb 27 20:27:21 2022 +0530`
	
	`6th commit.`
	
	`diff --git a/file6.txt b/file6.txt`
	`:...skipping...`
	`tag V-1.1.0`
	`Tagger: shaileshyadav7771 <shaileshyadav7958@gmail.com>`
	`Date: Sun Feb 27 20:29:42 2022 +0530`

	Release 1.1.0 commit b0163556e5591469dfa2788fa6f53449922867e3 (HEAD -> master, tag: v-1.1.0) Author: shaileshyadav7771 <shaileshyadav7958@gmail.com> Date: Sun Feb 27 20:27:21 2022 +0530 6th commit. **diff --git a/file6.txt b/file6.txt** **new file mode 100644** **index 0000000..603cb1b** **--- /dev/null** **+++ b/file6.txt** @@ -0,0 +1 @@ +First Line //Implemented as object.

So, because of above extra info It is highly recommended to use annotated Tag 😊.

```
/d/tagging (master)
$ git tag -l
v-1.0.0
v-1.1.0
```

Sir simple Tag is of text type and annotated Tag is of object type.

Proof:

```
/d/tagging (master)
$ git tag -v v-1.0.0
error: v-1.0.0: cannot verify a non-tag object of type commit.
```

For annotated Tag:

```
/d/tagging (master)
$ git tag -v v-1.1.0
object b0163556e5591469dfa2788fa6f53449922867e3
type commit
tag v-1.1.0
tagger shaileshyadav7771 <shaileshyadav7958@gmail.com> 1645973982 +0530

Release 1.1.0
error: no signature found
```

where this Tags will be stored?

Simple Tag	Annotated Tag
1. It will be stored at ***.git/refs/tags*** /d/tagging/.git/refs/tags (GIT_DIR!) $ cat v-1.0.0 58189bab9b92820fcba9e8215e3acaf390fe62e8 **Note:** Above one is the commit ID.	1. It will be stored at ***.git/refs/tags*** And at ***.git/objects*** /d/tagging/.git/refs/tags (GIT_DIR!) $ cat v-1.1.0 d7d34cc7e6f49dcad9aa17e1fd46188fef050fd8 **Note:** Above one is not the commit ID It is the **hash** of the Tag object 😊. **To check Type of an object we can use one command is there.**

```
/d/tagging/.git/refs/tags (GIT_DIR!)
$ git cat-file -t
d7d34cc7e6f49dcad9aa17e1fd46188fef050fd
8
tag
```

To check this Tag object pointing to which commit ID we can use -p

```
$ git cat-file -p
d7d34cc7e6f49dcad9aa17e1fd46188fef050fd
8
object
b0163556e5591469dfa2788fa6f53449922867e
3
type commit
tag v-1.1.0
tagger shaileshyadav7771
<shaileshyadav7958@gmail.com>
1645973982 +0530

Release 1.1.0
```

Now to check details in .git/objects

```
/d/tagging/.git (GIT_DIR!)
$ cd objects/

/d/tagging/.git/objects (GIT_DIR!)
$ ls
05/  49/  60/  7d/  b0/  c1/  d7/  info/
3e/  58/  77/  93/  bc/  c4/  eb/  pack/
```

Now check our HASH code `d7d34cc7e6f49dcad9aa17e1fd46188fef050fd8`
So, starting digit is d7 so check d7 folder 😊.

```
/d/tagging/.git/objects/d7 (GIT_DIR!)
$ ls
d34cc7e6f49dcad9aa17e1fd46188fef050fd8
```

Note: It is object right if we will check It contain binary data.

```
/d/tagging/.git/objects/d7 (GIT_DIR!)
$ cat d34cc7e6f49dcad9aa17e1fd46188fef050fd8
xU▒▒
▒O▒a▒y▒%▒d▒
          ▒▒
            .▒O▒▒EZ*}{+▒▒~_▒▒▒▒▒m
▒▒'▒J▒"▒▒▒▒s'▒#▒s9&▒▒lO▒f▒簫▒VX▒▒▒▒▒ ▒▒I+▒▒▒?1▒k?▒jG.▒1'▒▒▒ђ▒▒▒tRY~▒▒0{
```

Summary:

Light weight	Annotated Tag
1. git tag <tagname> 2. It is text label and not implemented as object.	1. git tag -a <tag-name> -m <tag-message> 2. It is internally Implemented as an Object.

3. Stored: ***.git.refs/tags*** folder 4. It won't store any extra information.	3. Stored at : ***.git/refs/tags*** & ***.git/objects*** 4. It will store extra information like tagger name, date of creation, message etc..

Chap16:Tag to previous commit and updating Tags.

In this chap we will discuss how to Tag a previous commit. (For more details, please refer previous chapter).

```
/d/tagging/.git/objects/d7 (GIT_DIR!)
$ git log --oneline
b016355 (HEAD -> master, tag: V-1.1.0) 6th commit.
58189ba (tag: V-1.0.0) 5th commit.
77ef69b 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

Sir as shown above suppose I forgot to create a Tag for 4th commit 😊 so I want to define Tag for it.

Syntax:

```
$ git tag -a <Tag-name> <for-which commit ID> -m "message"
```

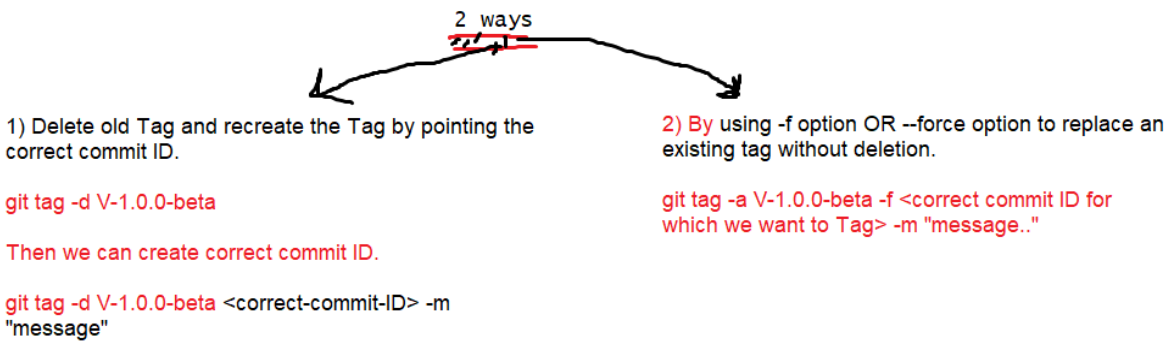
Note: we have used same command for annotated Tag here we are mentioning commit ID for which we want to create Tag. (If we will not mention commit ID then It'll be created for the latest commit 😊).

```
/d/tagging/.git/objects/d7 (GIT_DIR!)
$ git tag -a V-1.0.0-beta 77ef69b -m "Release 1.0.0 Beta version(Not for the
production use)"
```

```
/d/tagging/.git/objects/d7 (GIT_DIR!)
$ git log --oneline
b016355 (HEAD -> master, tag: V-1.1.0) 6th commit.
58189ba (tag: V-1.0.0) 5th commit.
77ef69b (tag: V-1.0.0-beta) 4th commit.
bce3b9d 3rd commit.
937f9ba second commit.
c162d32 first commit.
```

Updating Tags:

Now sir Suppose I have updated wrong commit ID to V1.0.0-beta so I need to update it how we can do that?




```
/d/tagging/.git/objects/d7 (GIT_DIR!)
$ git tag -a V-1.0.0-beta -f 937f9ba -m "Updating V-1.0.0-beta to second commit
from 4th commit.."
Updated tag 'V-1.0.0-beta' (was ae8780f)
/d/tagging/.git/objects/d7 (GIT_DIR!)
```

```
$ git log --oneline
b016355 (HEAD -> master, tag: V-1.1.0) 6th commit.
58189ba (tag: V-1.0.0) 5th commit.
77ef69b 4th commit.
bce3b9d 3rd commit.
937f9ba (tag: V-1.0.0-beta) second commit.
c162d32 first commit.
```

Perfect now we can see that it (Tag) is pointing to second commit 😊.

Que: For the same commit It is possible to use multiple Tag or not?
Ans: Yes 😊, Tag is just reference so we can use multiple references.

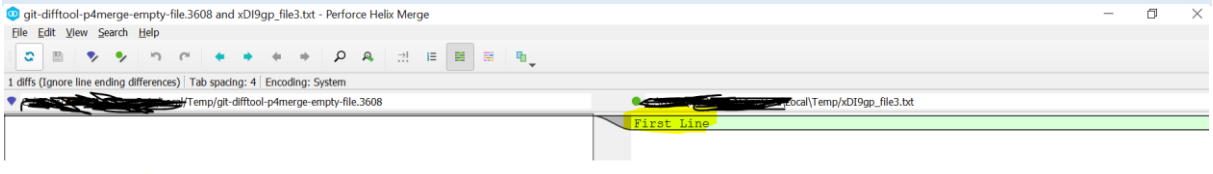
Que: for multiple commits it is possible to use same Tag?
Ans: No Sir, it should be unique within the repository.

Note: We know git diff commitID1 commitID2 command so same way instead of commit-ID we can use Tag name also.
e.g.: git diff V-1.0.0 V-1.1.0

even we can use diff-tool also.

```
$ git diff V-1.0.0-beta V-1.0.0
diff --git a/file3.txt b/file3.txt
new file mode 100644
index 0000000..603cb1b
--- /dev/null
+++ b/file3.txt
@@ -0,0 +1 @@
+First Line
diff --git a/file4.txt b/file4.txt
new file mode 100644
index 0000000..603cb1b
--- /dev/null
+++ b/file4.txt
@@ -0,0 +1 @@
+First Line
diff --git a/file5.txt b/file5.txt
new file mode 100644
index 0000000..603cb1b
--- /dev/null
+++ b/file5.txt
@@ -0,0 +1 @@
+First Line
```

```
/d/tagging (master)
$ git difftool V-1.0.0-beta V-1.0.0
```



Chap17: How to push Tag to remote Repository.

Topic we covered:
Define tag for previous commit
How to update existing tag without deleting that tag: -f option
How to compare tags

Now in this chap we will see how to push tags to the remote repository:
So for that if we will use

git push origin master

It will push code + commit history but not tags

We have to push tags separately:

Practical:

```
/d
$ mkdir tagging-push

/d
$ cd tagging-push/

/d/tagging-push
$ git init
Initialized empty Git repository in D:/tagging-push/.git/

/d/tagging-push (master)
$ echo "First Line" > abc.txt

/d/tagging-push (master)
$ git add .;git commit -m "first-commit."
[master (root-commit) 6da0398] first-commit.
 1 file changed, 1 insertion(+)
 create mode 100644 abc.txt

/d/tagging-push (master)
$ git log --oneline
6da0398 (HEAD -> master) first-commit.
```

//so above commit contain 1 file and 1 line in it. Let me go with same file and I do changes in it.

```
/d/tagging-push (master)
$ echo "second line" >> abc.txt

/d/tagging-push (master)
$ git add .;git commit -m "second-commit."
warning: LF will be replaced by CRLF in abc.txt.
The file will have its original line endings in your working directory
[master 6c75f34] second-commit.
 1 file changed, 1 insertion(+)
```

//so now second commit 1 file and 2 lines of data 😊.

```
/d/tagging-push (master)
$ echo "3rd line" >> abc.txt

/d/tagging-push (master)
$ git add .;git commit -m "3rd-commit."
[master 024ace7] 3rd-commit.
 1 file changed, 1 insertion(+)
```

//3rd commit one file and 3 lines.

```
/d/tagging-push (master)
$ echo "4th line" >> abc.txt

/d/tagging-push (master)
$ git add .;git commit -m "4th-commit."
[master a4b27d1] 4th-commit.
 1 file changed, 1 insertion(+)
```

//4th commit only 1 file abc.txt and 4 lines of data.

```
/d/tagging-push (master)
$ cat abc.txt
First Line
second line
3rd line
4th line

/d/tagging-push (master)
$ git log --oneline
a4b27d1 (HEAD -> master) 4th-commit.
024ace7 3rd-commit.
6c75f34 second-commit.
6da0398 first-commit.
```

Note: We already know that we can assign a tag to previous commit so let do it.

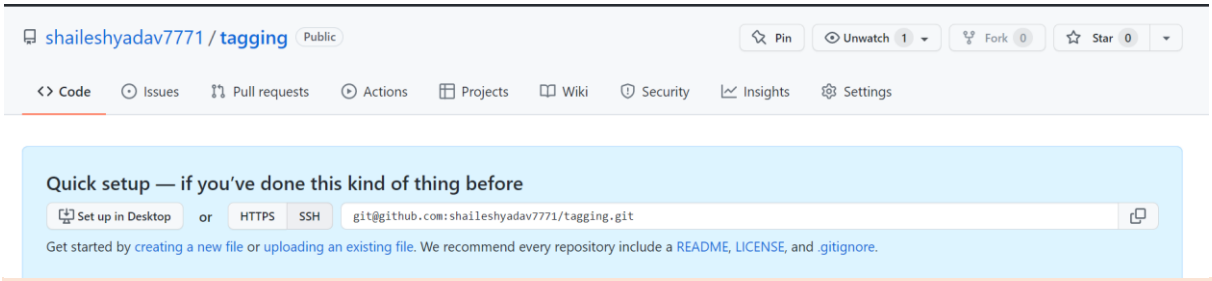
```
/d/tagging-push (master)
$ git tag -a V-1.0.0 6da0398 -m "Release 1.0.0"

/d/tagging-push (master)
$ git log --oneline
a4b27d1 (HEAD -> master) 4th-commit.
024ace7 3rd-commit.
6c75f34 second-commit.
6da0398 (tag: V-1.0.0) first-commit.
```

//now above we have defined annotated tag and let us define for another commit as well.

```
/d/tagging-push (master)
$ git tag -a V-1.1.0 6c75f34 -m "Release 1.1.0"

/d/tagging-push (master)
$ git tag -a V-1.2.0 024ace7 -m "Release 1.2.0"

/d/tagging-push (master)
$ git tag -a V-1.3.0 a4b27d1 -m "Release 1.3.0"

/d/tagging-push (master)
$ git log --oneline
a4b27d1 (HEAD -> master, tag: V-1.3.0) 4th-commit.
024ace7 (tag: V-1.2.0) 3rd-commit.
6c75f34 (tag: V-1.1.0) second-commit.
6da0398 (tag: V-1.0.0) first-commit.
```

Now I want to push this code to the remote repository.

So, in remote repository I have created one repo and we know before accessing we need to configure remote repo 😊



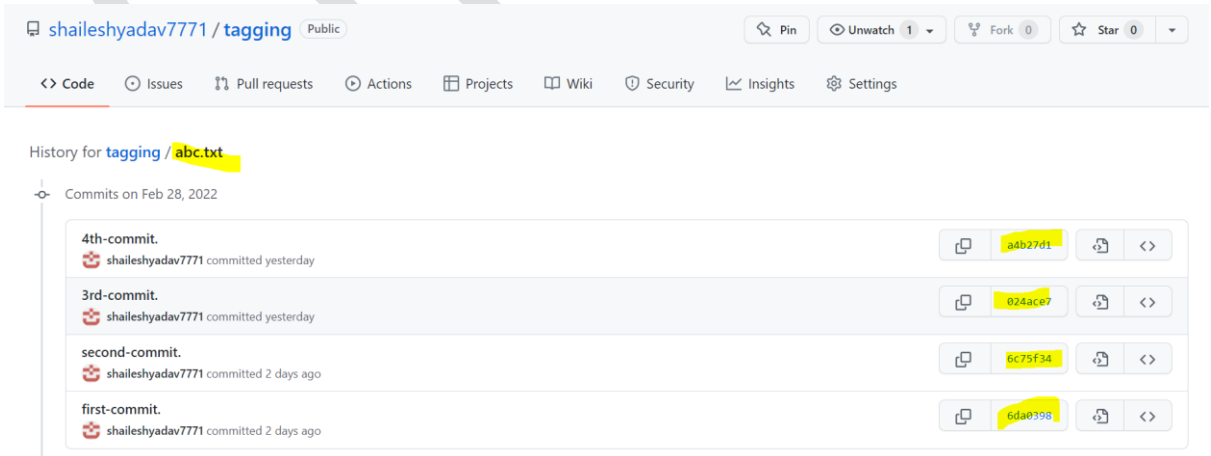

```
/d/tagging-push (master)
$ git remote -v


/d/tagging-push (master)
$ git remote add origin git@github.com:shaileshyadav7771/tagging.git

/d/tagging-push (master)
$ git remote -v
origin  git@github.com:shaileshyadav7771/tagging.git (fetch)
origin  git@github.com:shaileshyadav7771/tagging.git (push)
```

now remote repo is added so we can perform fetch and push operation 😊.

```
/d/tagging-push (master)
$ git push origin master
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (12/12), 925 bytes | 308.00 KiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:shaileshyadav7771/tagging.git
 * [new branch]      master -> master
```

Please note It will push all the data including commit history (so one file but 4 commit).

Que: Sir Tags moved or not?
Ans: We have already discussed that normal push command will not send tagging to remote repository.



Proof:

How to push single tag?

Command:

```
$ git push origin master <tagname>
```

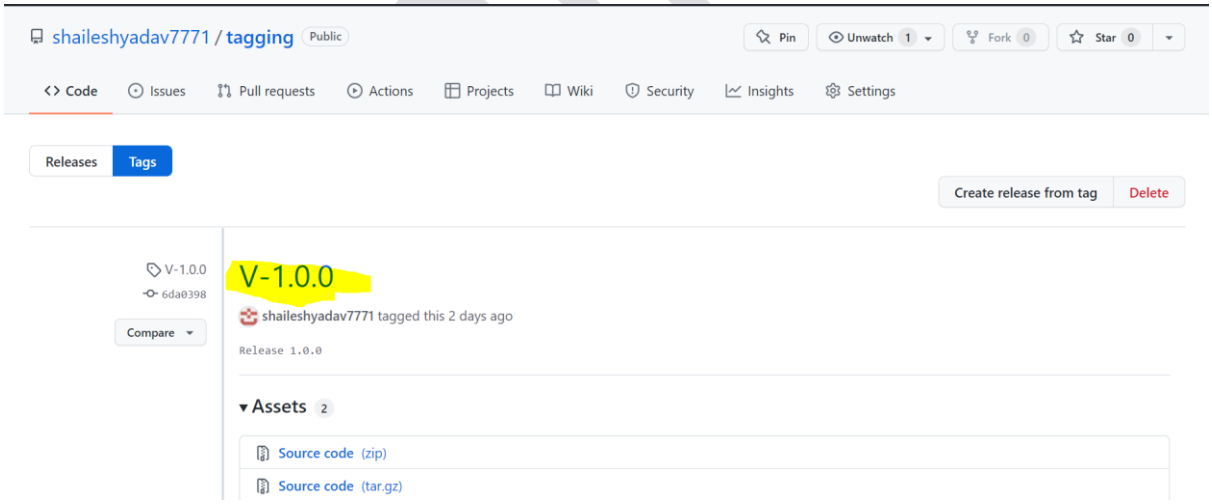
Let me check no of tags we have.

```
/d/tagging-push (master)
$ git log --oneline
a4b27d1 (HEAD -> master, tag: V-1.3.0, origin/master) 4th-commit.
024ace7 (tag: V-1.2.0) 3rd-commit.
6c75f34 (tag: V-1.1.0) second-commit.
6da0398 (tag: V-1.0.0) first-commit.
```

We have 4 tags.
Suppose I have to push only one Tag. Let me take v-1.0.0

```
/d/tagging-push (master)
$ git push origin V-1.0.0     #we have not used master but recommended.
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 170 bytes | 170.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:shaileshyadav7771/tagging.git
 * [new tag]         V-1.0.0 -> V-1.0.0
```

Proof:
Generally, tags we define for release purpose (so in UI we can see it under release).



Note: Sir till V-1.0.0 the file contains only one-line 😊 (It is related with first commit).
So, we can see the **zip** for window env and **.gz** for the Linux env so happily we can download it and see the source code.

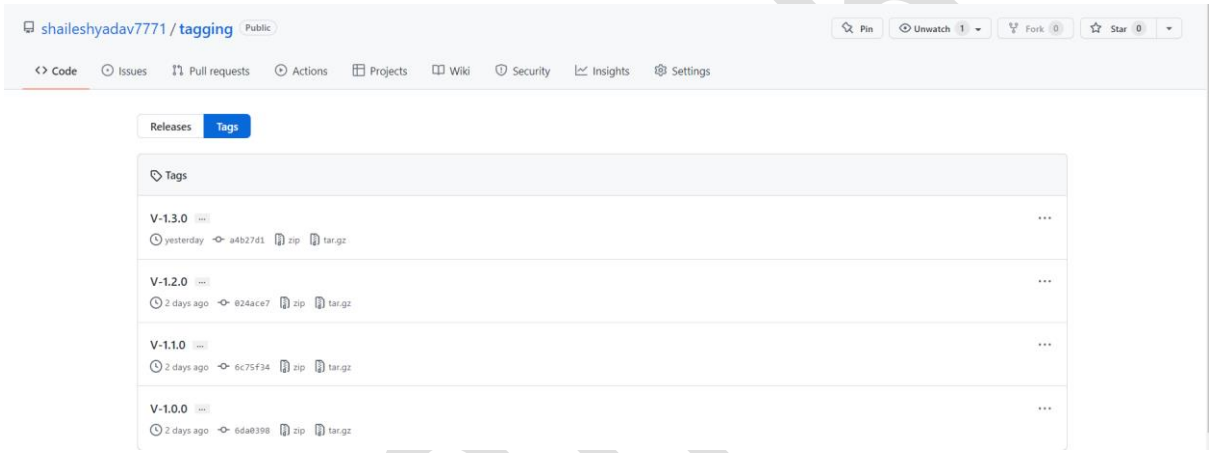
Note: In code although we are having 4 commits (all code) but till now we have only one release 😊.

How to push all the tags?

```
$ git push origin master --tags
```
//push all the tags to remote repository

```
/d/tagging-push (master)
$ git push origin master --tags
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 446 bytes | 446.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:shaileshyadav7771/tagging.git
 * [new tag]         V-1.1.0 -> V-1.1.0
 * [new tag]         V-1.2.0 -> V-1.2.0
 * [new tag]         V-1.3.0 -> V-1.3.0
```

Let check,



Note: latest tag V-1.3.0 will contain 4 lines (all the data 😊).

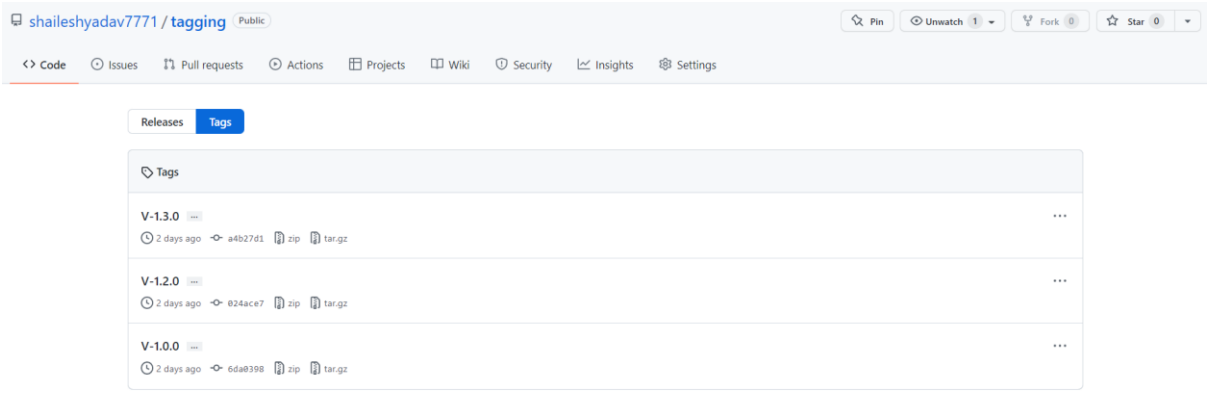
How to delete Tags from remote repository?

We have already discussed how to delete tag from local repository and command for that is `$ git tag -d V-1.1.0` but for remote repository It is different.

We need to use

```
$ git push origin :V-1.1.0
```

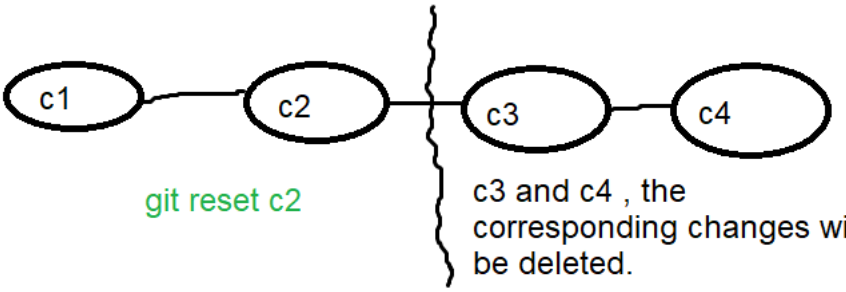
```
/d/tagging-push (master)
$ git push origin :V-1.1.0
To github.com:shaileshyadav7771/tagging.git
 - [deleted]         V-1.1.0
```



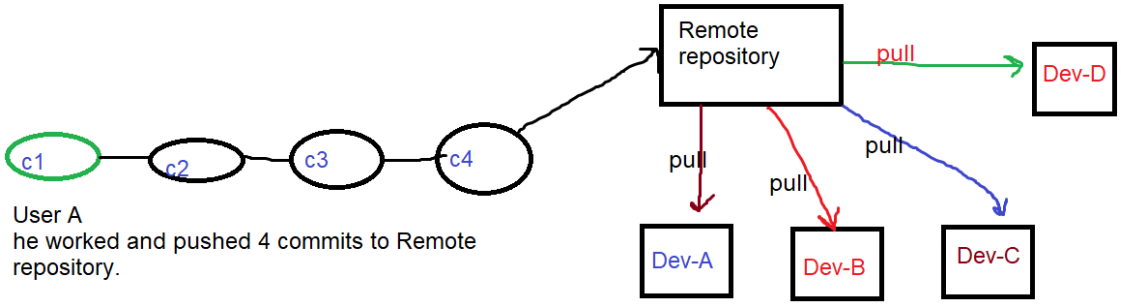
Note: It will not delete it from local repo we need to delete it if we want 😊.

Chap18: git revert command.

We have already discussed about git reset command (please refer notes).
git reset –mixed
git reset --soft
git reset –hard

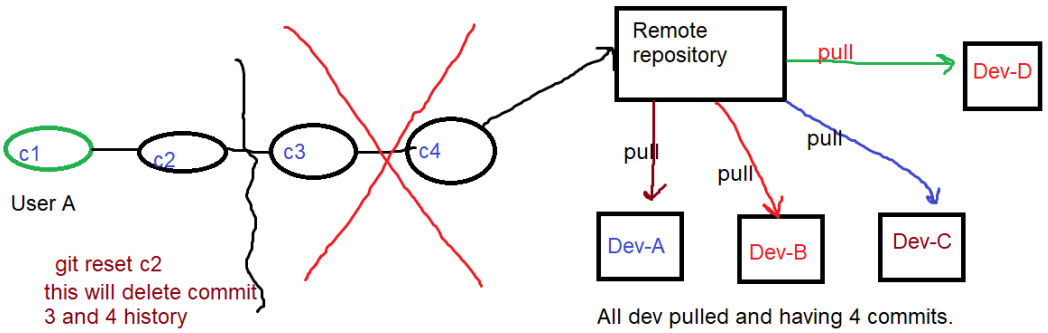


Note: git reset is destructive command and not recommended to use it on public repository.
For example, suppose **user A** is working and he have 4 commits in his local repository, and he pushed 4 commits to remote repo.



step1

So, remote repository having 4 commits and now Dev A, B, C, D have pulled code from remote repo. And suppose in between User A have used git reset c2 command which will delete c3, c4 commit from his local repository and if he Pushto remote repo then there will be conflict because other's Dev A, B, C, D are having 4 commits whereas user A is having 2 commits.

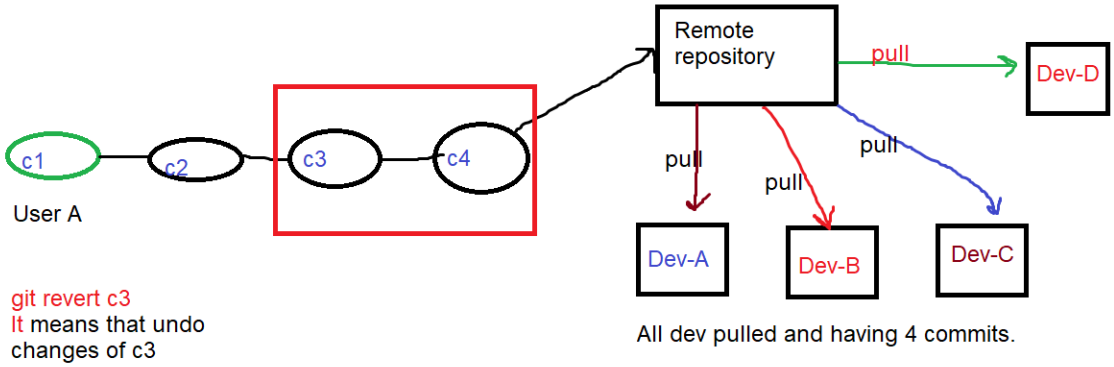


step2:
when UserA push code then there will be conflict as It will have 2 commit wheras other Dev are having 4 commit.(so reset is not recommonded in remote repo related work).

So, it is always recommended to use git reset local and not on remote repo (It will delete commit history).

Note: So, to overcome above problem happily we can use git revert command and It will not delete commit history 😊. It reverts the required commit by creating a new commit i.e It will undo a particular commit without deleting commit history.

In case of **revert**:



without deleting commit history of c3 and c4 and new commit will be created.(In our case with c2 a new commit will be created).
So including c1,c2,c3,c4 a new commit **c2n** will be also there/created.

Note: git reset delete where revert will not delete anything and this will not affect other developer.

c2 and new created **c2n** will have same changes.
Note: if we want c2 then we need to revert c3.

Practical:

```
/d
$ mkdir revertdemo

/d/revertdemo
$ git init
Initialized empty Git repository in D:/revertdemo/.git/
/d/revertdemo (master)
```

```
$ echo "First line " > file1.txt

/d/revertdemo (master)
$ git add .;git commit -m "first commit in master-UserA"
[master (root-commit) 19d239a] first commit in master-UserA
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
```

Now let me add in same file and create commit 😊. (Note we know file1.txt is already present in staging area so we can skip add and can directly commit with -am).

```
/d/revertdemo (master)
$ echo "second line" >> file1.txt;git commit -am "second commit in master-
[master 345a7dc] second commit in master-UserA
 1 file changed, 1 insertion(+)


/d/revertdemo (master)
$ echo "third line" >> file1.txt;git commit -am "third commit in master-UserA"
[master 5f57d9d] third commit in master-UserA
 1 file changed, 1 insertion(+)

/d/revertdemo (master)
$ echo "fourth line" >> file1.txt;git commit -am "4th commit in master-UserA"
[master d0c3eb0] 4th commit in master-UserA
 1 file changed, 1 insertion(+)
```

Let us verify

```
/d/revertdemo (master)
$ cat file1.txt
First line
second line
third line
fourth line

/d/revertdemo (master)
$ git log --oneline --graph
* d0c3eb0 (HEAD -> master) 4th commit in master-UserA
* 5f57d9d third commit in master-UserA
* 345a7dc second commit in master-UserA
* 19d239a first commit in master-UserA
```

Note: in first commit file1.txt contain 1 line and in file2.txt file contain 2 lines and file3.txt contain 3 lines and file4.txt contain 4 lines.

So now to test suppose I want to revert c4 commit then what happen.

Simply sir c4 content will be undo (commit history will not be deleted 😊) and with c3 and new commit will be created.

Now after using revert C4 changes will be undo and c3 changes will be preserved and a new commit will be created.

```
/d/revertdemo (master)
$ git revert d0c3eb0
[master e993e58] Revert "4th commit in master-UserA"
 1 file changed, 1 deletion(-)
```

Note: After entering above revert command It will ask for commit message (As shown in below Image). I have not modified it (default commit message).

Now let us verify It 😊.

```
/d/revertdemo (master)
$ git log --oneline
e993e58 (HEAD -> master) Revert "4th commit in master-UserA"
d0c3eb0 4th commit in master-UserA
5f57d9d third commit in master-UserA
345a7dc second commit in master-UserA
19d239a first commit in master-UserA
```

Now we will have 5 commits.

Now how many lines will be there in the file1.txt (before using revert command It was 4 lines) 😊.

```
/d/revertdemo (master)
$ cat file1.txt
First line
second line
third line
```

this 2 are having same content.

```
$ git log --oneline
e993e58 (HEAD -> master) Revert "4th commit in master-UserA"
d0c3eb0 4th commit in master-UserA
5f57d9d third commit in master-UserA
345a7dc second commit in master-UserA
19d239a first commit in master-UserA
```

Proof:

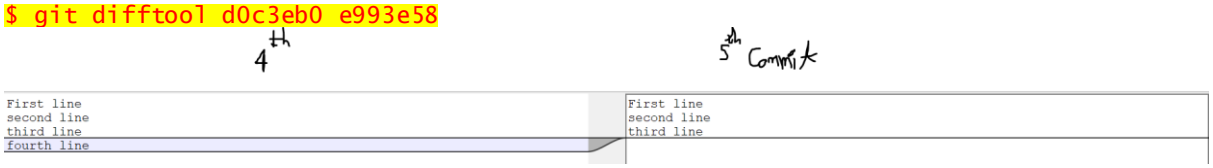
```
/D/revertdemo (master)
$ git diff e993e58 5f57d9d
```

Note: if we compare it with 4th commit vs 5th commit, we know 4th commit (4 lines) but 5th commit only 3 lines.

```
$ git diff d0c3eb0 e993e58
diff --git a/file1.txt b/file1.txt
index 5af0d07..0bbb242 100644
--- a/file1.txt
+++ b/file1.txt
@@ -1,4 +1,3 @@
 First line
```

```
 second line
 third line
-fourth line
```

```
$ git difftool d0c3eb0 e993e58
```



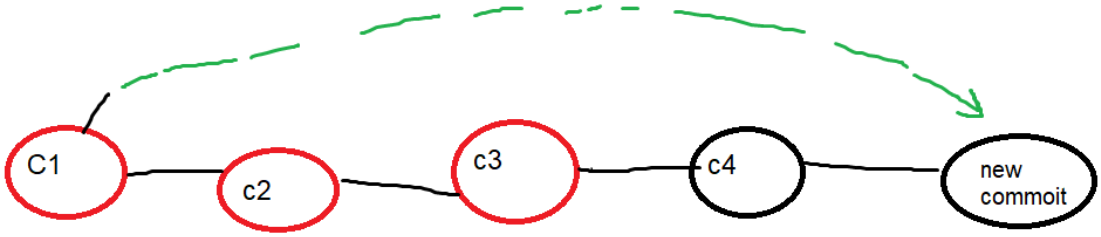
How to revert specific commit (in above we have reverted specific commit 😊).
Yes, we can revert it

Practical:

```
/D
$ mkdir revertdemo2

/D (master)
$ cd revertdemo2/

/D/revertdemo2 (master)
$ echo "First line " > file1.txt

/D/revertdemo2 (master)
$ git add .;git commit -m "first commit in master-UserA"
[master (root-commit) c165c18] first commit in master-UserA
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt

/D/revertdemo2 (master)
$ echo "second line" >> file1.txt;git commit -am "second commit"
[master e58009d] second commit
 1 file changed, 1 insertion(+)

/D/revertdemo2 (master)
$ echo "3rd line" >> file1.txt;git commit -am "3rd commit"
[master 9898622] 3rd commit
 1 file changed, 1 insertion(+)

/D/revertdemo2 (master)
$ echo "4th line" >> file1.txt;git commit -am "4th commit"
[master cf44244] 4th commit
 1 file changed, 1 insertion(+)
```

So total 4 commits.

```
/D/revertdemo2 (master)
$ git log --oneline
cf44244 (HEAD -> master) 4th commit
9898622 3rd commit
e58009d second commit
c165c18 first commit in master-UserA
```



git revert c2

which stage will be preserved?
C1
so with C1 (contain one line) a new commit will be created.
with c1(one line).there may be chance of merge because c4 contain 4 lines wheras c1 contain 1 lines (same file right so it'll ask which data we want).

Note: Before this we discussed there we were reverting most recent commit but here c1 and moost recent commit both having different no of lines.

```
/D/revertdemo2 (master)
$ git revert e58009d
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
error: could not revert e58009d... second commit
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>", then run
hint: "git revert --continue".
hint: You can instead skip this commit with "git revert --skip".
hint: To abort and get back to the state before "git revert",
hint: run "git revert --abort".
```

```
/D/revertdemo2 (master|REVERTING)
$ vi file1.txt
First line
<<<<<<< HEAD
second line
3rd line
4th line
=======
>>>>>>> parent of e58009d (second commit)
```

//so, from above we can see first commit contain 1 line and 4th commit contain 4 line which we want?
So, let us remove all line and keep First line.

```
/D/revertdemo2 (master|REVERTING)
$ git add file1.txt;git commit -m "reverted second commit"
[master 392ccf4] reverted second commit
 1 file changed, 3 deletions(-)
```

So now we will have 5 commits (1 we added as a part of reverting c2 😊).

```
/D/revertdemo2 (master)
$ git log --oneline --graph
* 392ccf4 (HEAD -> master) reverted second commit
* cf44244 4th commit
* 9898622 3rd commit
* e58009d second commit
* c165c18 first commit in master-UserA
```

```
/D/revertdemo2 (master)
```

```
$ git diff 392ccf4 cf44244
diff --git a/file1.txt b/file1.txt
index f0aa640..9b75b15 100644
--- a/file1.txt
+++ b/file1.txt
@@ -1 +1,4 @@
 First line
+second line
+3rd line
+4th line
```

Note: If we are having multiple revert commit so best way to use git revert -n option (OR –no commit option we need to use) means no commit will be created and once done then create one final commit😊).

https://github.com/shaileshyadav7771/remote_repo/commit/392ccf42176fbaf25c465502defeea5095394473

Chap19: Cherry picking (git cherry-pick command).

It is Advanced Topic in GIT.
So, first question is what is cherry 😊?

As shown in image cheery picking means in general, we will pick one cherry and eat it.

Similarly in GIT also we will select specified commit from child branch to merge with master branch.

(For example, suppose my master branch contains 4 commits and I have used git checkout -b feature branch Now my feature branch will also contain 4 commits.
Now I have worked on some requirement as a part of that I have added 5 commits to my child (feature branch)).

Now feature branch is having total 9 commit but my master is having 4 commits. So, one way we know is change to master and then use `$ git merge feature` (this need to be run on master branch because master need additional changes from child) but this will merge all the changes in child (like commit 5,6,7,8,9 all will be available to the master branch).

But what if my requirement is that I need only 7 commit OR let say 9th commit of child to be available to my master branch 😊.
So, cherry can help here (cherry means random picking).

Note: If we merge feature branch to the master branch, then all commits, and the corresponding changes will be added to the master branch. Instead of all commits, we can pick an arbitrary commit of feature branch and we can append that commit to the master branch. It is possible by using cherry-pick command.

Advantages:

1. Code sharing so we can pickup any commit and merge it with master and push it to remote (no need to merge all commit).
2. We can use for bug hot fixes. Assume that dev A has started working on a new feature. During that new feature development, he found/identified that there is pre-existing bug in master branch. So here developer can use cherry pick concept and he will fix that issue by creating one commit in feature branch and then cherry pick up in master branch (before It affect other users).

Practical:

```
/D
$ mkdir cherrypick

/D
$ cd cherrypick/
```

```
/D/cherrypick
$ git init
Initialized empty Git repository in D:/cherrypick/.git/

/D/cherrypick (master)
$ echo "First line by master" > file1.txt

/D/cherrypick (master)
$ git add .;git commit -m "first commt by master."
[master (root-commit) 8901495] first commt by master.
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt

/D/cherrypick (master)
$ echo "second line by master" >> file1.txt;git commit -am "second commit by master
branch."
[master b47ed3b] second commit by master branch.
 1 file changed, 1 insertion(+)

/D/cherrypick (master)
$ git log --oneline
b47ed3b (HEAD -> master) second commit by master branch.
8901495 first commt by master.

/D/cherrypick (master)
$ cat file1.txt
First line by master
second line by master
```

Note: Now I want to develop new feature for above code so let me create new branch.

```
/D/cherrypick (master)
$ git checkout -b feature
Switched to a new branch 'feature'
```

We know for feature branch all master details will be available.

```
/D/cherrypick (feature)
$ echo "3rd line by master" >> file1.txt;git commit -am "3rd commit by feature
branch."
[feature 3d530cb] 3rd commit by feature branch.
 1 file changed, 1 insertion(+)

/D/cherrypick (feature)
$ echo "4th line by master-BUG Fixing" >> file1.txt;git commit -am "4th commit by
feature branch."
[feature 83c9d7d] 4th commit by feature branch.
 1 file changed, 1 insertion(+)
```

So total 4 commits are there in the feature branch.

```
/D/cherrypick (feature)
$ git log --oneline
83c9d7d (HEAD -> feature) 4th commit by feature branch.
3d530cb 3rd commit by feature branch.
b47ed3b (master) second commit by master branch.
8901495 first commt by master.

/D/cherrypick (feature)
$ cat file1.txt
First line by master
second line by master
3rd line by master
4th line by master-BUG Fixing
```

Now we want to merge a particular commit from feature to master branch and that concept is called as cherry pick.
So first we need to switch to master branch.

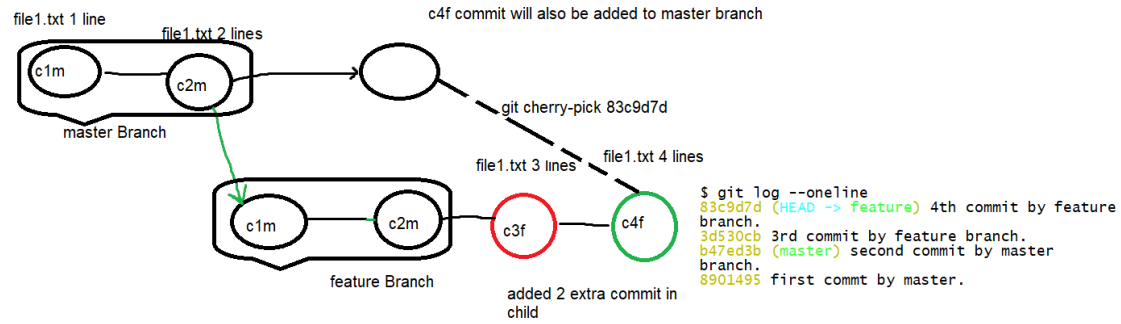
```
/D/cherrypick (feature)
$ git checkout master
Switched to branch 'master'
```

So, it will have only 2 commits right.

```
/D/cherrypick (master)
$ git log --oneline
b47ed3b (HEAD -> master) second commit by master branch.
8901495 first commt by master.
```

And if we will open file1.txt then we will see only 2 lines will be there

```
/D/cherrypick (master)
$ cat file1.txt
First line by master
second line by master
```



Note: Now we want to merge a particular commit of feature branch to master branch.

Now suppose I want only 4 th commit of feature so we need to use
git cherry-pick 83c9d7d (from master branch)

```
/D/cherrypick (master)
$ git cherry-pick 83c9d7d
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
error: could not apply 83c9d7d... 4th commit by feature branch.
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>", then run
hint: "git cherry-pick --continue".
hint: You can instead skip this commit with "git cherry-pick --skip".
hint: To abort and get back to the state before "git cherry-pick",
hint: run "git cherry-pick --abort".
```

```
/D/cherrypick (master|CHERRY-PICKING)
$ vi file1.txt
```

```
First line by master
second line by master
<<<<<<< HEAD
=======
3rd line by master
4th line by master-BUG Fixing
>>>>>>> 83c9d7d (4th commit by feature branch.)
~
~
~
~
```

So let us keep first line and second line. (Delete other line esc + dd).

```
/D/cherrypick (master|CHERRY-PICKING)
Date: Sat Mar 12 12:38:25 2022 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .file1.txt.swp

/D/cherrypick (master)
$ git log --oneline
d29aba1 (HEAD -> master) git cherry trying for latest commit of feature.
b47ed3b second commit by master branch.
8901495 first commt by master.
```

Now suppose I want to git cherry pick 3rd commit of child branch then.

```
/D/cherrypick (master)
$ git cherry-pick 3d530cb
[master 9a1f3b7] 3rd commit by feature branch.
 Date: Sat Mar 12 12:37:03 2022 +0530
 1 file changed, 1 insertion(+)
```

So now new commit will be created, and 3rd commit details will be available to master branch.
(Before this we have tried for the latest commit and that is why It raise conflict message)

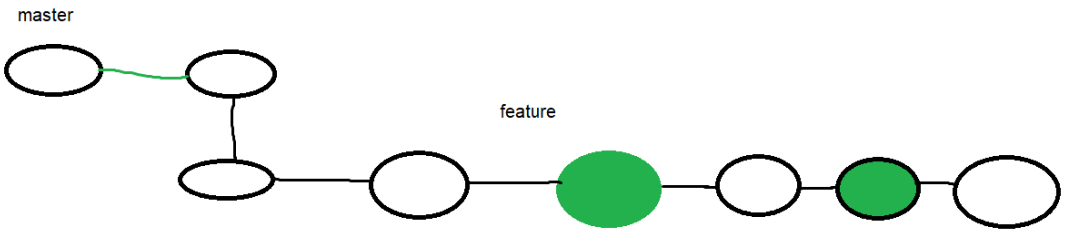
```
/D/cherrypick (master)
$ git log --oneline
9a1f3b7 (HEAD -> master) 3rd commit by feature branch.
d29aba1 (origin/master) git cherry trying for latest commit of feature.
b47ed3b second commit by master branch.
8901495 first commt by master.
```

Note: Because of this now BUG resolve (means added solution code) and if we will check our master branch file1.txt will contain 3 lines (3rd line is added as a part of cherry-pick).
https://github.com/shaileshyadav7771/git-cherry-pick/blob/master/file1.txt

Note: commit ID in feature and in master after cherry-pick new commit object creation will change (because in master new commit is created with child branch changes).

```
/D/cherrypick (master)
$ cat file1.txt
First line by master
second line by master
3rd line by master
```

So, 2 lines from earlier and 3 lines by cherry-pick commit 😊.
(If any conflict we need to resolve and commit manually).



Note: Sie as shown above if we have multiple commit in featuer branch and use cherry-pick command then multiple commit will be created for example in above case consider 3,5 commit so we want to avoid creating commit in master branch we can use below command.

```
$ git cherry-pick --no-commit commit_ID
```

//changes will be available in mastre branch but commit ID won't be created.

```
At last when work iscompleted then manually create commit
```

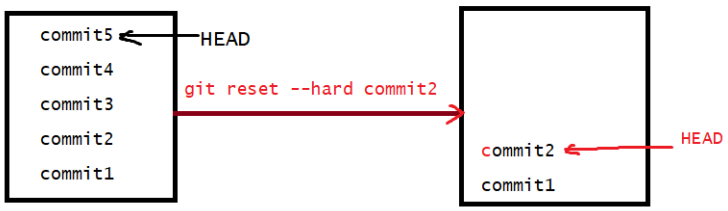
Note: Cherry pick is faster compared to merge (because here we are talking about picking specific commit).
Cherry-pick may cause duplicate commit and hence in most of the scenario we can use merge operation instead of *cherry-pick*.

More Details:

```
$ git help cherry-pick
```

Chap20: git reflog command

Sir we know git log but what is this git reflog 😊.
It means reference log.



Note: We know above git reset --hard command It will delete all other commit and we can see after delte now HEAD is pointing to the commit 2.
If we will check git log we will see only 2 commits.

Sir, while learning we have discussed `$ git reset --hard` is very dangerous command and it will delete commit from local repo + staging area + and working directory 😊.

So, by using above git ***reflog*** command we can undo/recover changes for above such destructive commands.

Note: Sir whatever operation we are performing in local repository for that git ***reflog*** is creating/maintain a separate records. (***reflog*** entries will be available by default for the **90** days 😊).

> `$ git reflog` => To log(show/display) all git operation whatever we performed on our local repository.

we can use `Git reflog` command to know only local repository operations/changes but not remote repository operations.

//It is like Linux history command. If we want to know details w.r.t branch we need to use like below.

> `$ git reflog <branch_name>` => It will show all git operation performed on local repository related to particular branch.

Practical:

```
/D
$ mkdir reflog

/D
$ cd reflog/

/D/reflog
$ git init
Initialized empty Git repository in D:/reflog/.git/

/D/reflog (master)
$ echo "first line ">>file1.txt
/D/reflog (master)
```

```
$ git add .;git commit -m "first commit."
[master (root-commit) dd68871] first commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt

/D/reflog (master)
$ echo "first line" > file2.txt

/D/reflog (master)
$ git add .;git commit -m "second commit."
[master dded2a3] second commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt


/D/reflog (master)
$ echo "first line" > file3.txt

/D/reflog (master)
$ git add .;git commit -m "3rd commit."
[master 5dd6b3d] 3rd commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file3.txt


/D/reflog (master)
$ echo "first line" > file4.txt

/D/reflog (master)
$ git add .;git commit -m "4th commit."
[master e5ed829] 4th commit.
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt
```

So, we have created 4 files and 4 commits.

```
/D/reflog (master)
$ ls
file1.txt  file2.txt  file3.txt  file4.txt
```

Now if we will use git reflog command.

```
/D/reflog (master)
$ git reflog
e5ed829 (HEAD -> master) HEAD@{0}: commit: 4th commit.
5dd6b3d HEAD@{1}: commit: 3rd commit.
dded2a3 HEAD@{2}: commit: second commit.
dd68871 HEAD@{3}: commit (initial): first commit.
```

//It look like log messages but contain some extra info 😊.

Now suppose if I am using `$ git reset dd68871` so it will delete all other commit ID except first commit.

```
/D/reflog (master)
$ git reset --hard dd68871
HEAD is now at dd68871 first commit.

/D/reflog (master)
$ ls
file1.txt
```

//above we can see that after hard reset we are having only one file,

Alright now if we check log, we will see only one commit whereas ***reflog*** will have all the commit details 😊.

`/D/reflog (master)`	//whereas `/D/reflog (master)`

$ git log --oneline
dd68871 (HEAD -> master) first commit.
	$ git reflog
	dd68871 (HEAD -> master) HEAD@{0}: reset: moving to dd68871
	e5ed829 HEAD@{1}: commit: 4th commit.
	5dd6b3d HEAD@{2}: commit: 3rd commit.
	dded2a3 HEAD@{3}: commit: second commit.
	dd68871 (HEAD -> master) HEAD@{4}: commit (initial): first commit.
	
	//we can see complete history available.

Now I want to restore changes.

```
/D/reflog (master)
$ git reset --hard e5ed829
HEAD is now at e5ed829 4th commit.

/D/reflog (master)
$ git log --oneline
e5ed829 (HEAD -> master) 4th commit.
5dd6b3d 3rd commit.
dded2a3 second commit.
dd68871 first commit.
```

So, we can see that all 4-commit recovered.

```
/D/reflog (master)
$ ls
file1.txt  file2.txt  file3.txt  file4.txt
```

Note: If we open ***git reflog*** we can see all operation what we have performed.

Note: Now we are having only one branch (so content will be same).

```
/D/reflog (master)
$ git reflog master
e5ed829 (HEAD -> master) master@{0}: reset: moving to e5ed829
dd68871 master@{1}: reset: moving to dd68871
e5ed829 (HEAD -> master) master@{2}: reset: moving to e5ed829
dd68871 master@{3}: reset: moving to dd68871
e5ed829 (HEAD -> master) master@{4}: commit: 4th commit.
5dd6b3d master@{5}: commit: 3rd commit.
dded2a3 master@{6}: commit: second commit.
dd68871 master@{7}: commit (initial): first commit.
```

Let us create a new branch and check it.

```
/D/reflog (master)
$ git checkout -b feature
Switched to a new branch 'feature'

/D/reflog (feature)
$ echo "first line" > x.txt

/D/reflog (feature)
$ git add .;git commit -m "first commit by feature branch."
[feature 22260f0] first commit by feature branch.
 1 file changed, 1 insertion(+)
 create mode 100644 x.txt

/D/reflog (feature)
$ echo "first line" > y.txt

/D/reflog (feature)
```

```
$ git add .;git commit -m "2nd commit by feature branch."
[feature b04c338] 2nd commit by feature branch.
 1 file changed, 1 insertion(+)
 create mode 100644 y.txt
```

```
/D/reflog (feature)
$ git log --oneline
b04c338 (HEAD -> feature) 2nd commit by feature branch.
22260f0 first commit by feature branch.
e5ed829 (master) 4th commit.
5dd6b3d 3rd commit.
dded2a3 second commit.
dd68871 first commit.
```

```
/D/reflog (feature)
$ git reflog
b04c338 (HEAD -> feature) HEAD@{0}: commit: 2nd commit by feature branch.
22260f0 HEAD@{1}: commit: first commit by feature branch.
e5ed829 (master) HEAD@{2}: checkout: moving from master to feature
e5ed829 (master) HEAD@{3}: reset: moving to e5ed829
dd68871 HEAD@{4}: reset: moving to dd68871
e5ed829 (master) HEAD@{5}: reset: moving to e5ed829
dd68871 HEAD@{6}: reset: moving to dd68871
dd68871 HEAD@{7}: reset: moving to dd68871
e5ed829 (master) HEAD@{8}: commit: 4th commit.
5dd6b3d HEAD@{9}: commit: 3rd commit.
dded2a3 HEAD@{10}: commit: second commit.
dd68871 HEAD@{11}: commit (initial): first commit.
```

Now suppose if I want to know about feature branch log.

```
/D/reflog (feature)
$ git reflog feature
b04c338 (HEAD -> feature) feature@{0}: commit: 2nd commit by feature branch.
22260f0 feature@{1}: commit: first commit by feature branch.
e5ed829 (master) feature@{2}: branch: Created from HEAD
```

Note: Whatever command we can use with git log (like git log –oneline, git log --stat) same we can use with ***git reflog(***`$ git reflog --stat`***)*** command as well 😊.