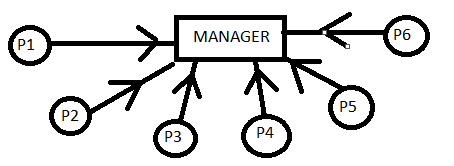
**Introduction to Git**

In previous time, when we work on a software development project we use to write different codes manually, merge that & compile.



It is very hard for a manager to remember who is giving which code.

So, to merge that codes we used a tool instead of person i.e. Software Configuration Management or Source Code Management. It is used to manage versions of code, giving versions to codes to remember file.

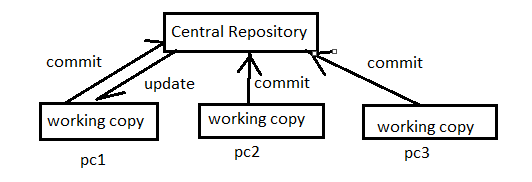
Types of source code management:

1-Centralised version control system.

2-Distributed Version control system.

**Centralized Control Version System(CVCS):**

Before git we use cvcs.



Architecture of CVCS

SVN tool is used in cvcs.

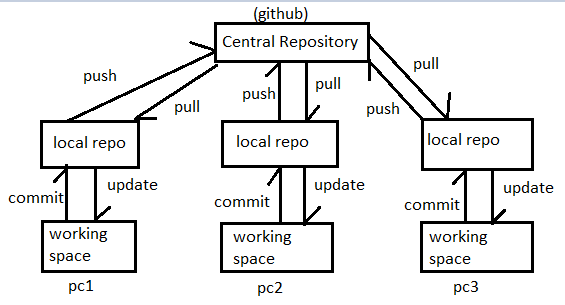
**Drawback of CVCS:**

* It’s not locally available, meaning you always need to be connected to a network to perform any action.
* Since everything is centralized, if central repository or server gets failed or crashed, you will loose entire data.

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

In DVCS, every contributor has a local copy or ‘clone’ of the central repository i.e. everyone maintain a local repository of their own which contains all the files & metadata present in main repository.

* Git is a DVCS.
* Git is manufactured by **Linus Torvald** in 2005.
* Before git linux uses bit keeper. It is a third party software. Linux uses bit keeper for versioning or version control.
* Linux stop using bit keeper because of some deal issue.
* Git is designed for linux kernel. It works on linux kernel.
* Git is a software or tool. It can be downloaded from internet.
* Github is different from git. It works on remote level.
* Git is a service work on local system level to manage versioning.



Architecture of DVCS

* In DVCS, we work on local system & save that work in local hard disk.
* Internet is not required to do any work because data will save in your system first.
* If repository(remote server) goes down or crashes, then no need to worry because your data will be saved in your local repository.
* Distributed means your code copy of same work is distributed at more than one place.
* Version control means, if you change your data very small then all teammates will know.

Difference between CVCS & DVCS:-

|  |  |
| --- | --- |
| CVCS | DVCS |
| In this, a client need to get local copy of source from server do the changes required & commit those changes to central source on server. | In DVCS, each client can have a local repository as well as have a complete history on it, client need to push the changes to branch which will be pushed then to server repository. |
| Cvcs system are easy to learn & setup. | Dvcs system are different for beginners. Multiple commands needs to be remembered. |
| Working on branches is difficult in cvcs. Developer often faces merge conflict. | Working on branches is easier in Dvcs. Developers faces less conflict. |
| Cvcs system don’t provide offline access. | Dvcs systems are working fine on offline mode as a client copies the entire repository on their local machine. |
| Cvcs is slower as every command needs to communicate with server. | Dvcs is faster as mostly user deals with local copy without lifting server every time. |
| If cvcs server is down developer can’t work. | If dvcs server is down, developer can work using their local copies. |
| Don’t have personal working copy. |  |

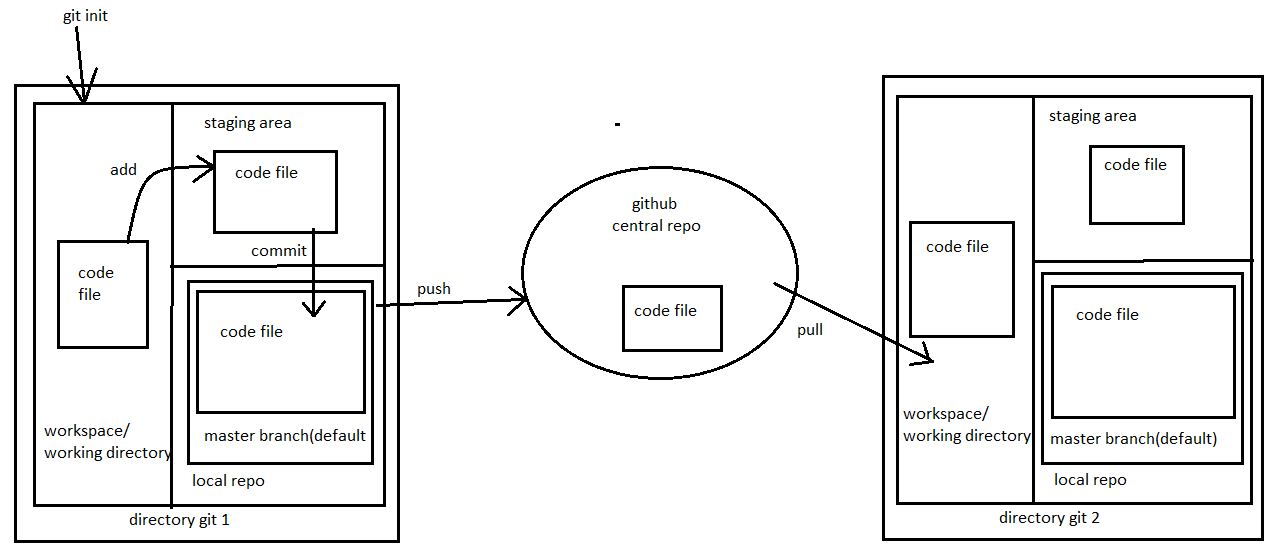
**Remember:**

|  |
| --- |
| * Before CVCS, local version control system was used. * It is used to save your codes on your local PC or Laptop, But if your PC or Laptop is crashed then all data will be lost. |

**Stages of Git & It’s Terminology**

**Git workflow with Github:**

|  |  |
| --- | --- |
| Stages | Description |
| 1 | Launch linux machine |
| 2 | Install git |
| 3 | Make a directory |
| 4 | Run git init(above directory converts into .git repository(local repo) |
| 5 | .git will divide into 3 region:  ->workspace/working directory  ->staging area  ->local repo |
| 6 | Coding is done into workspace |
| 7 | Add coding file in staging area |
| 8 | Now commit code file from staging area to local repo branch |
| 9 | Push coding file to central repo(github) |
| 10 | Pulling code to other machine from github, you can see data in local repo/staging area/working directory. |



Working process of git

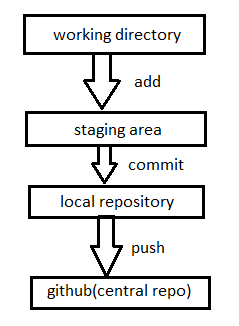
Repository:-

* It’s a place where you have all your codes or kind of folder in server.
* It’s a kind of folder related to one product.
* Changes are personal to that particular repository.

Server:- It stores all repository(local repo is also a server but it is private server). It contains metadata also.

Working Directory:-

* Where you see files physically & do modification.
* At a time you can work on particular branch.
* In CVCS, developers generally makes modification & commit their changes directly to the repository. But git uses different strategy, it doesn’t track each & every modified file. Whenever you do commit an operation, git looks for the files present in the staging area, only those files present in the staging area are considered for commit & not all the modified files.



Summary of git workflow

Commit:-

* It means sending code file from staging area to local repository.
* It stores changes in local repository you will get an commit-ID.
* It’s a 40 alpha-numeric characters.
* It uses SHA-1 checksum concept.
* Even if you change one dot(.) commit-ID will get changed.

SHA-1 Checksum Concept:-

* If you have a code of 1000 words in a file, so SHA-1 checksum will evaluate a binary value(like 234123411) & file will be sent to a person. Other side there will be SHA-1 checksum implemented & the output value for SHA-1 of that file is also same as sender value then it means there is no changes are made in that file, receiver gets that file as it is sent by sender.
* If any small changes is done like dot(.) then SHA-1 checksum value will be changed.
* It is used to know the status of code is changed or not in between sending & receiving.

Commit-ID/Version-ID/Version:-

* Reference to identity each change.
* To identify who changed the file.

Tags:- It assign a meaningful name with a specific version in the repository, once a tag is created for a particular save, even if you create anew commit, it will not be updated.

Snapshots(Increamental Backup):-

* Represents some data of a particular time.
* It is always incremental i.e. it stores the changed(appended data) only, not entire copy.
* It’ll save only updated stuff previous code will not be saved with new code line as a new file, but when you retrieve all code line it will automatically merge old & new code line Y show all code line in a single file.

Push:- Push operation copies the changes from a local repository server to a remote or central repo. It is used to store changes permanently into the git repository.

Pull:- Pull operation copies the changes from a remote repository to a local machine.

Branch:-

* Product is same, so one repository but different task .
* Each task has one separate branch.
* Finally merges all branches(code).
* Useful when you want to work parallel.
* Can create one branch on the basis of another branch.
* Changes are personal to that particular branch.
* Default branch is ‘Master’.
* File created in workspace will be visible in any of the branch workspace until you commit. Once you commit then that file belongs to that particular branch.

**Advantages of Git**

* Free & open source.
* Fast & Small:- As most of the operations are performed locally, therefore it’s fast. Small in size too.
* Security:- Git uses a common cryptographic hash function called secure hash function(SHA-1) to name & identify objects within it’s database.
* No need of powerful hardware.
* Easier Branching:- If we create a new branch, it’ll copy all the codes to the new branch.

**Types of Repositories:**

1. Bare Repository(Central Repo)

* Store & share only.
* All central repositories are bare repo.

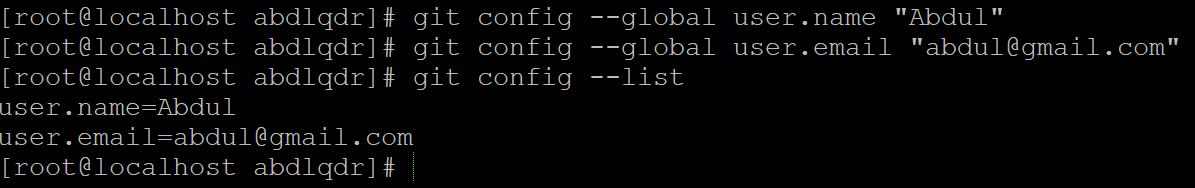
1. Non-Bare Repository(Local Repo)

* Where you can modify the files.
* All local repositories are non-bare repository.

**How to use git & create github account?**

Configuring git:-

* sudo su
* yum update –y
* yum install git
* git –version
* git config –global user.name “abdul”
* git config –global user.email [abdul@gmail.com](mailto:abdul@gmail.com)
* git config –list (to see created user in git)

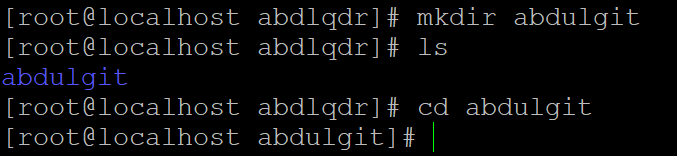


Creating github account:-

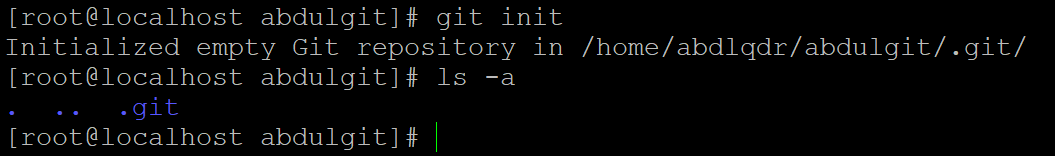
* open [www.github.com](http://www.github.com)
* create an account by sign up.
* Click join free plan.
* Now click on complete setup.
* Now verify your email from gmail.
* Sign out then sign in again.

**How to Commit, Push & Pull from Github?**

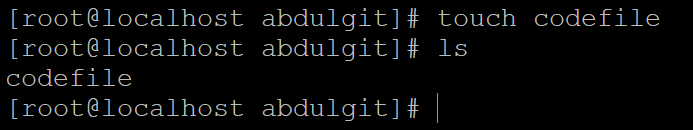
* Login into linux machine.
* Create a directory & go inside it.



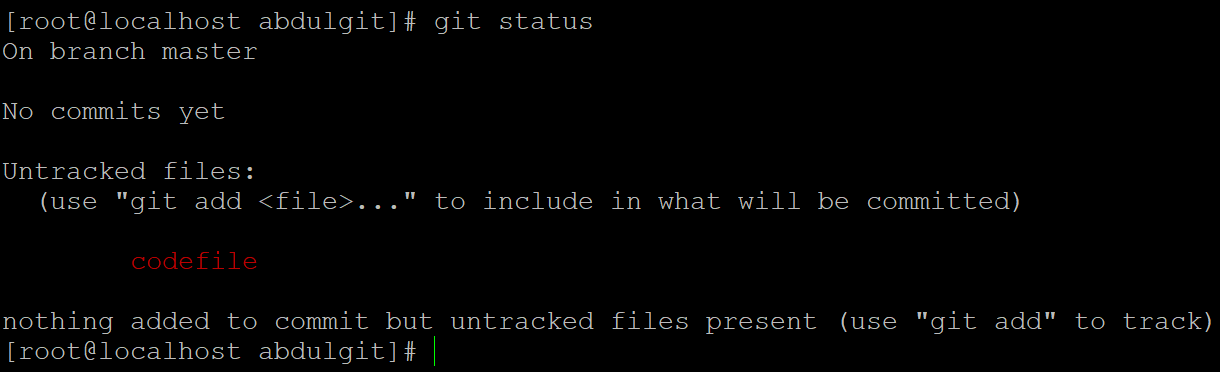
* Run (git init) command inside directory to make it local repo.



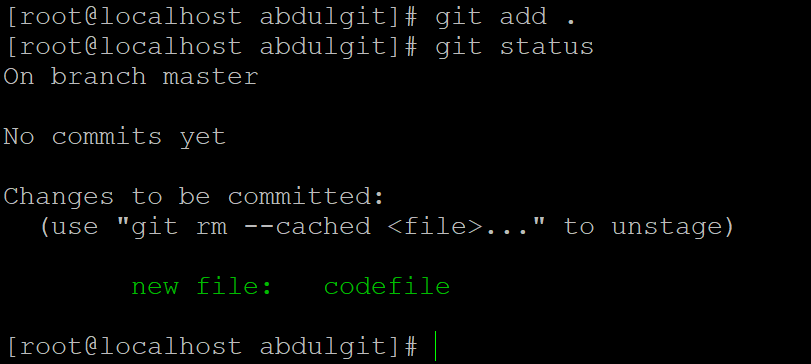
* Create a file-> touch myfile(put some data).



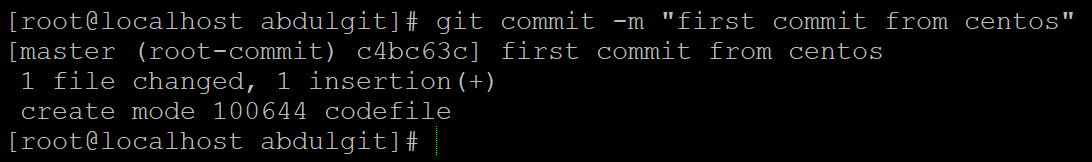
* git status (to check what data is in local repo).



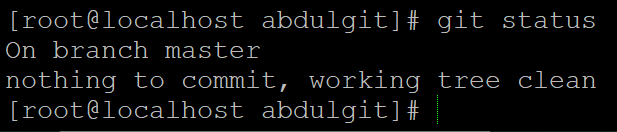
* git add . (to add code file from working area to staging area).



* git commit –m “first commit from centos” (to commit & give a message).



* git status (to check status of a code file).



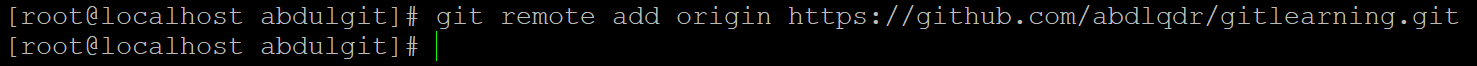
* git log (it tells which commit is done by which user).



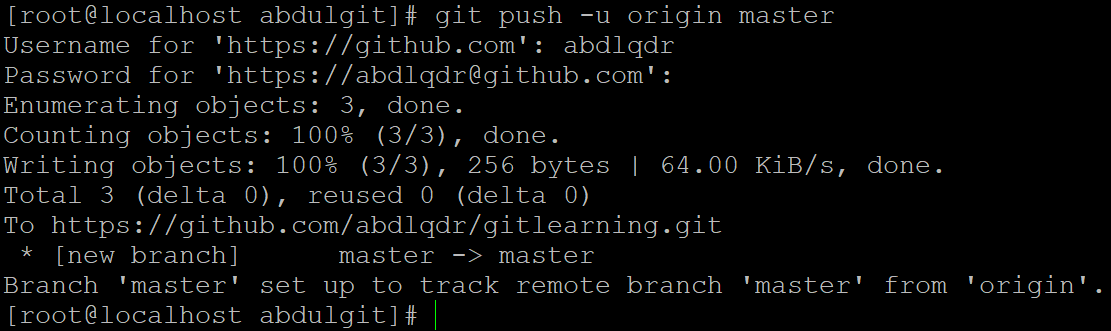
* git show <commit-id> (to see code of a particular commit).



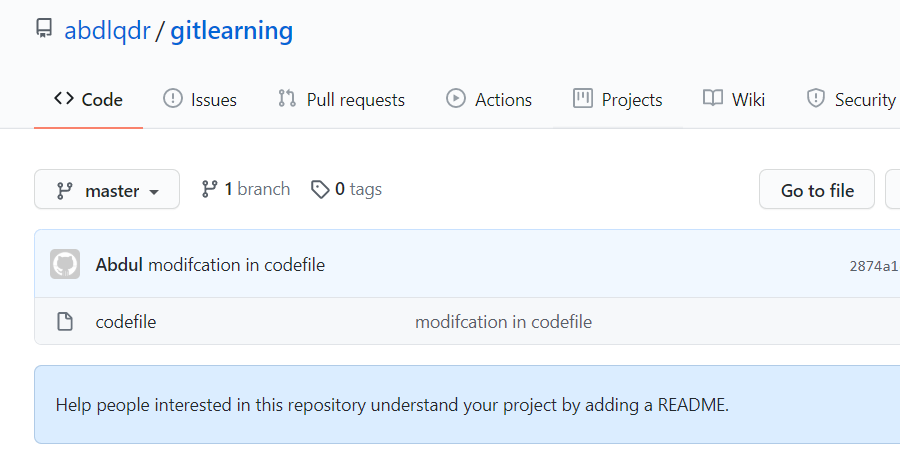
* git remote add origin <url> (to add github(central repo) with our local repo).



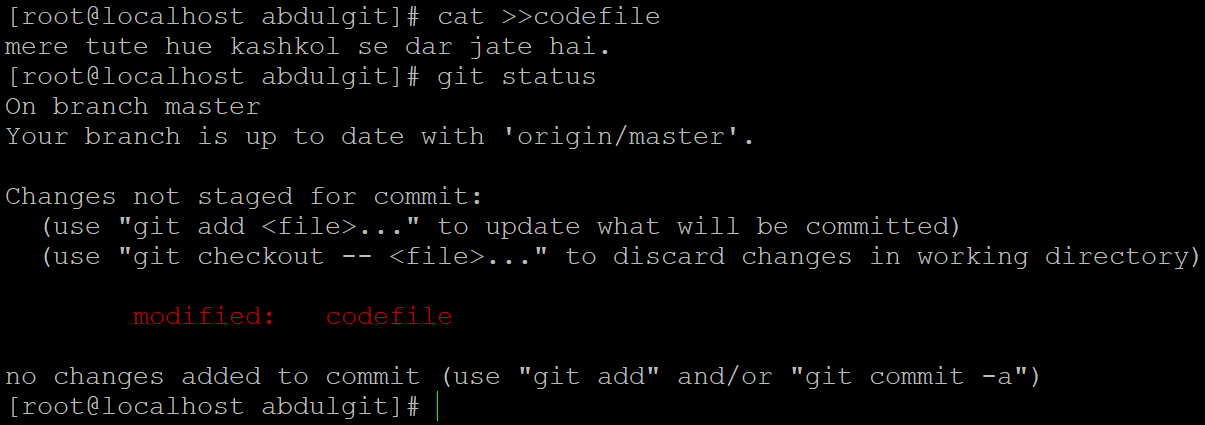
* git push –u origin master (to push code file from local repo to github in master branch).



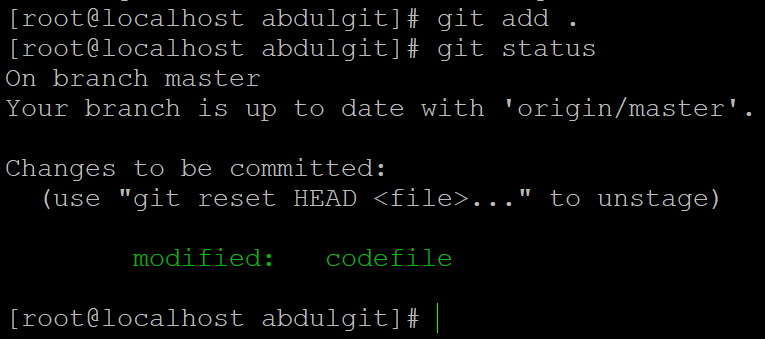
You can see your file in github(central repo).



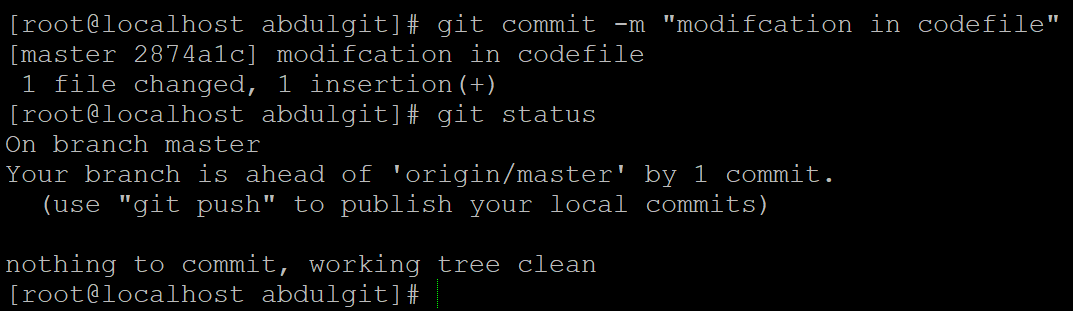
Now updating content in the previous file & checking status,



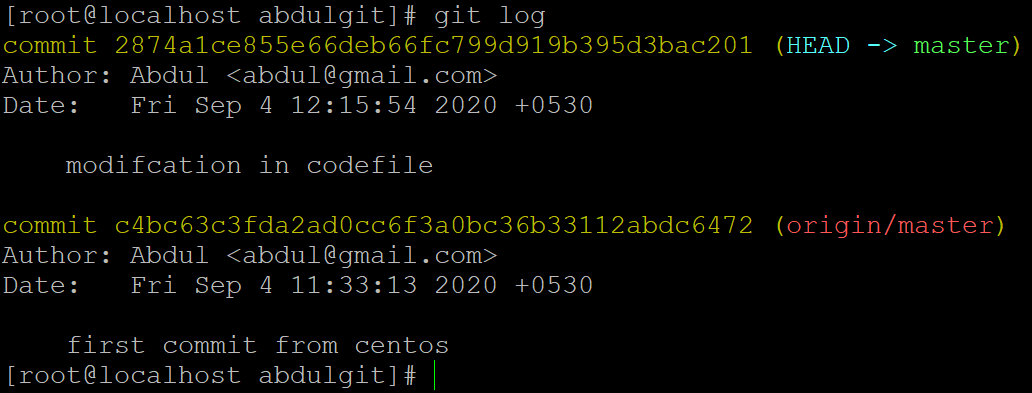
Now adding file to staging & checking status,



Now committing file to local repo & checking status.



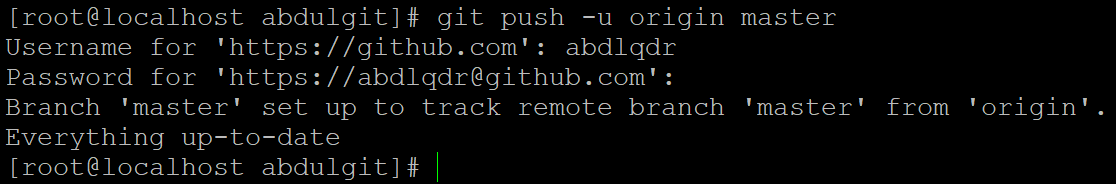
Now looking total git commits.



Now seeing the updated content in file.



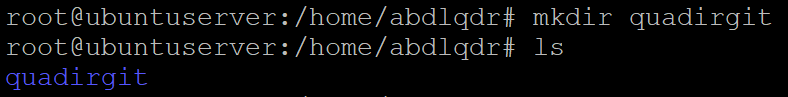
Now pushing modification code file to github.



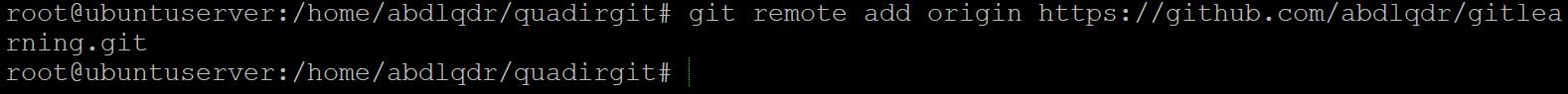
**Pulling Mechanism**

Login into linux machine.

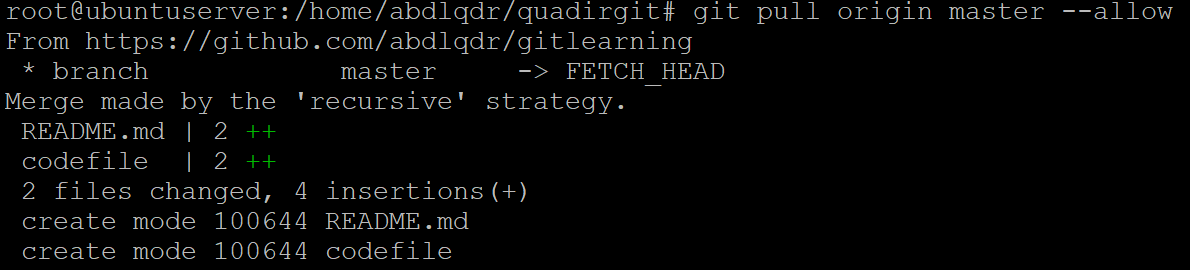
Create a directory, go inside it & run ‘git init’ command to create local repo.



Git remote add origin <github url> (to connect github with local repo).



git pull –u origin master/git pull origin master –allow

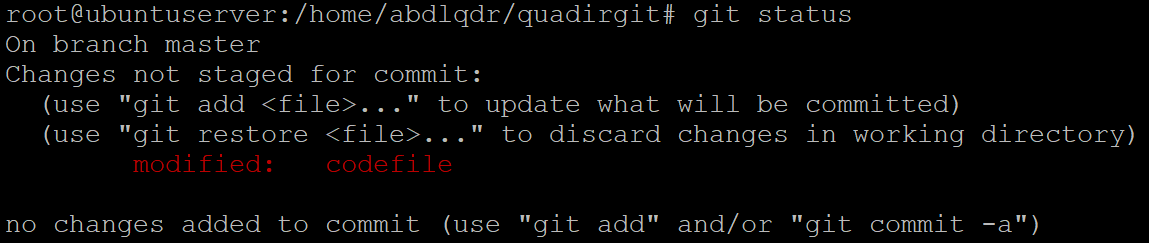


git log (to show all commit)

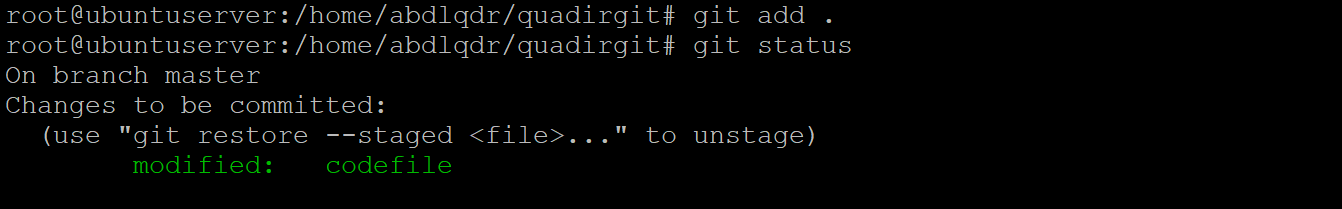
add some code to codefile



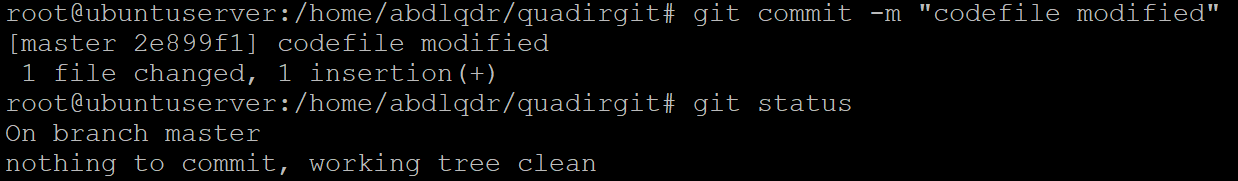
git status



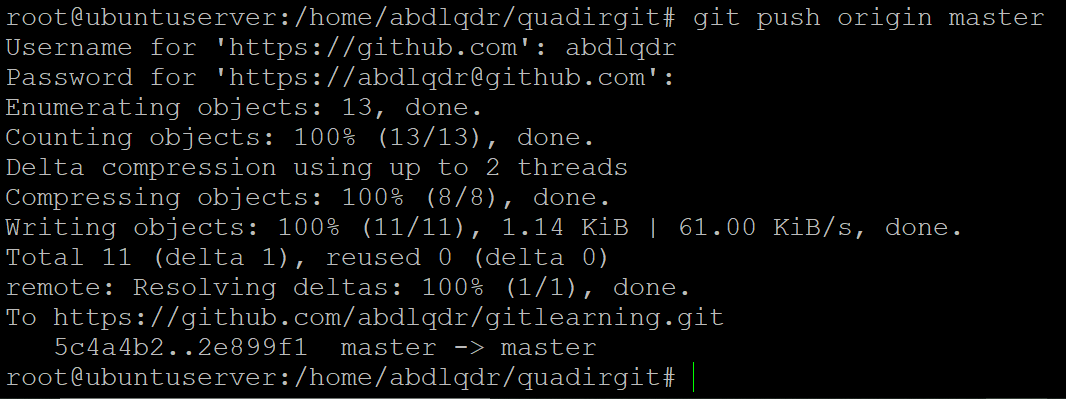
git add . then git status



git commit –m “codefile modified” /then git status

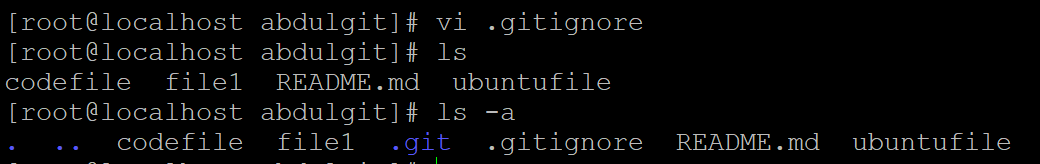


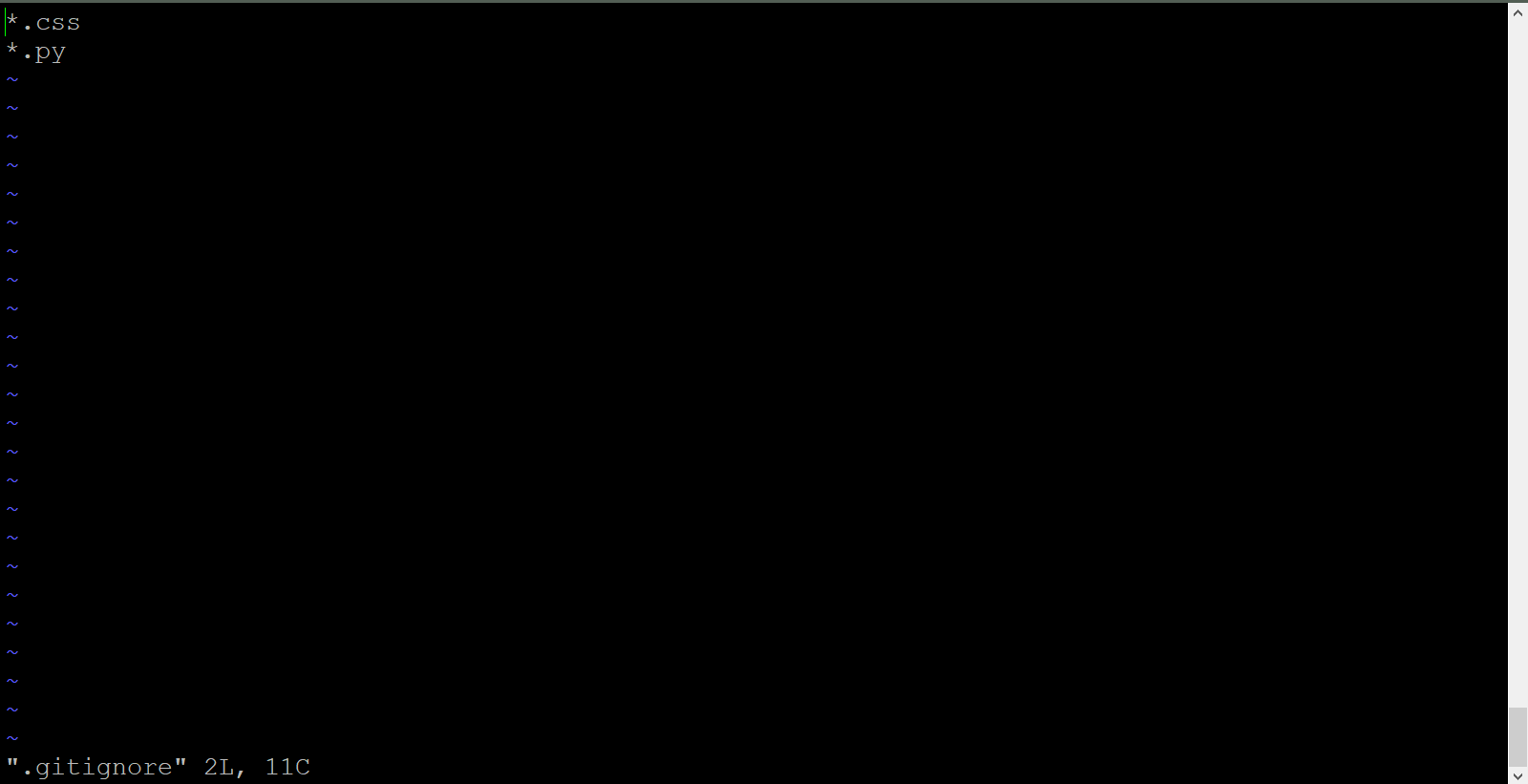
git push origin master



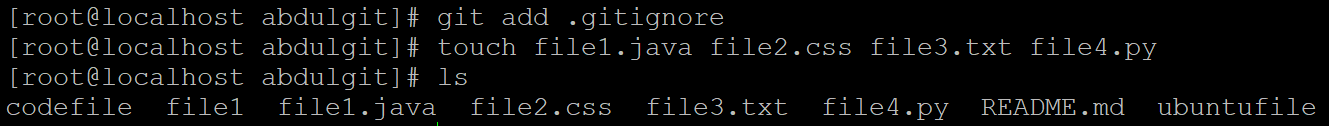
**git ignore Command**

* To ignore some files while committing.
* Create one hidden file “.gitignore” & enter file format which you want to ignore.

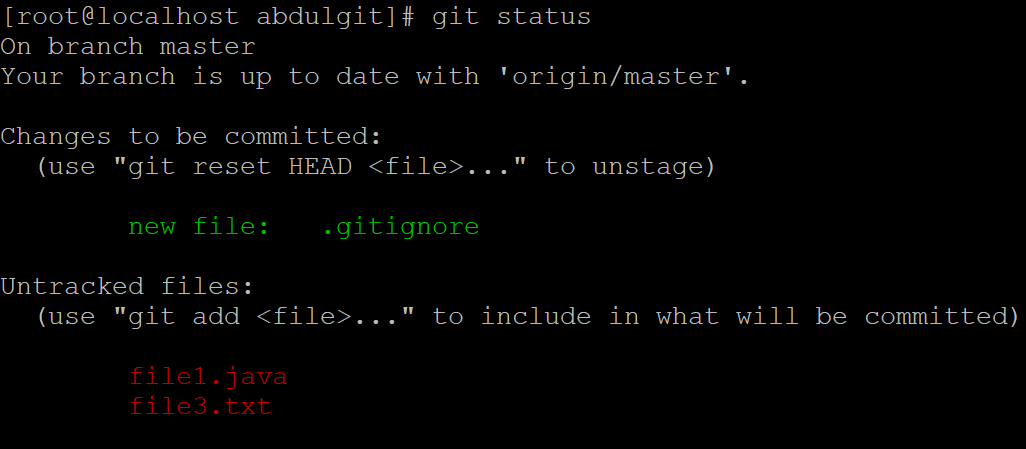




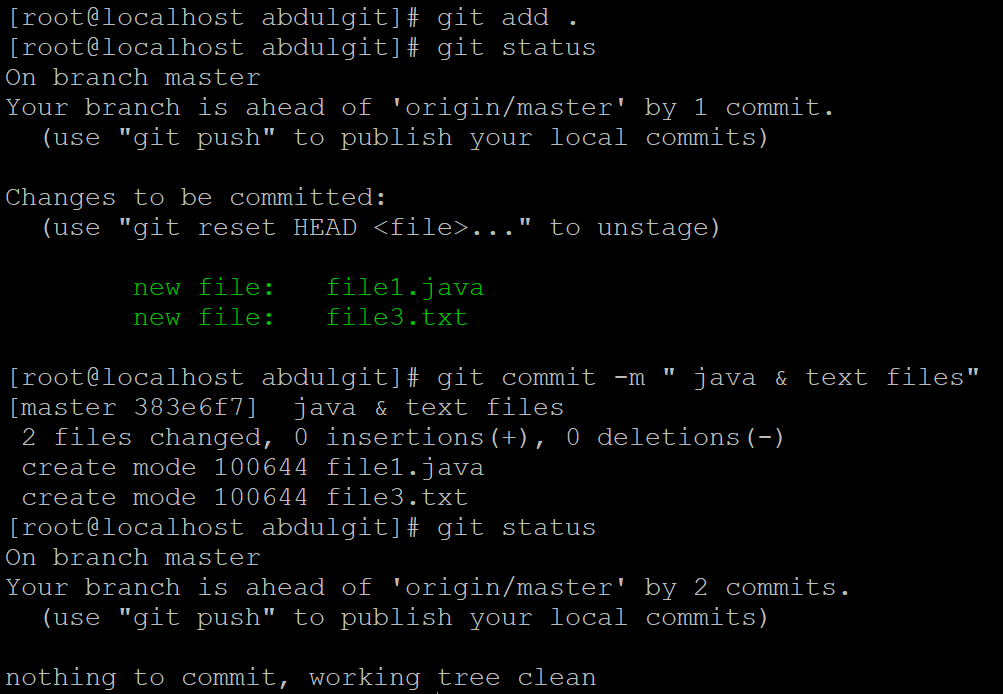
* Run git add .ignore & create some file using touch command.



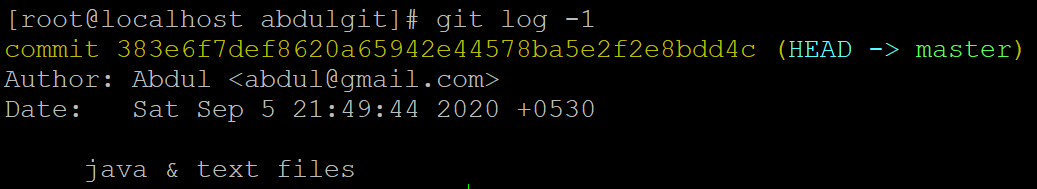
* Run git status, you will see it is showing only 2 files other 2 are ignored.



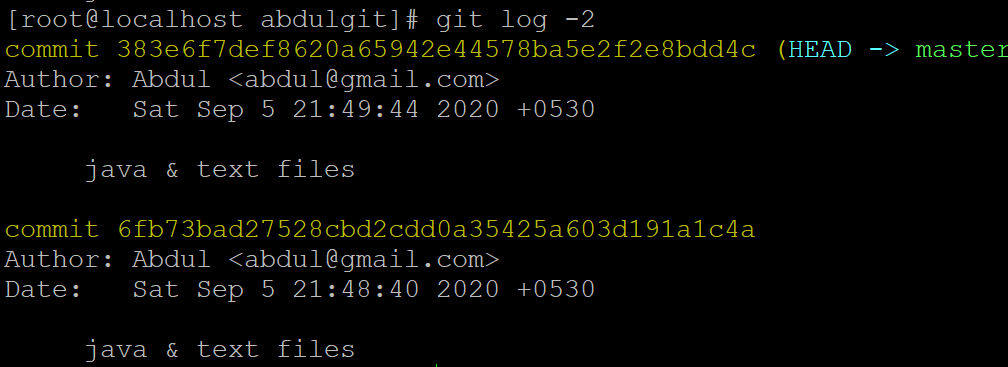
* Now, you can commit these files as you need.



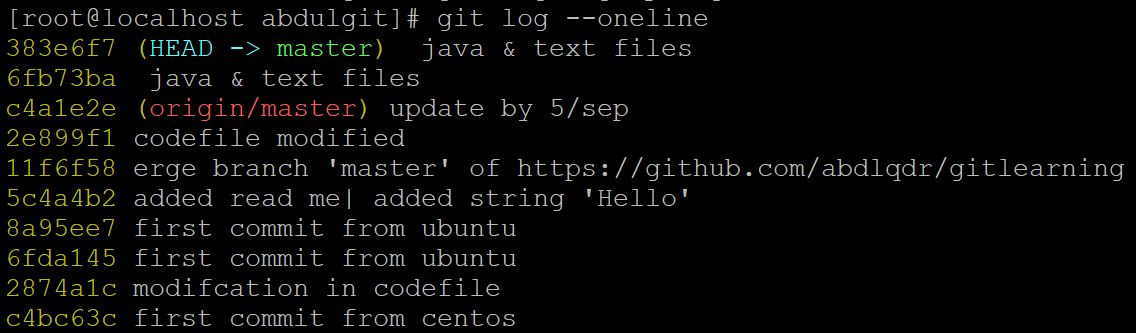
* git log -1 (to show last & recent commit).



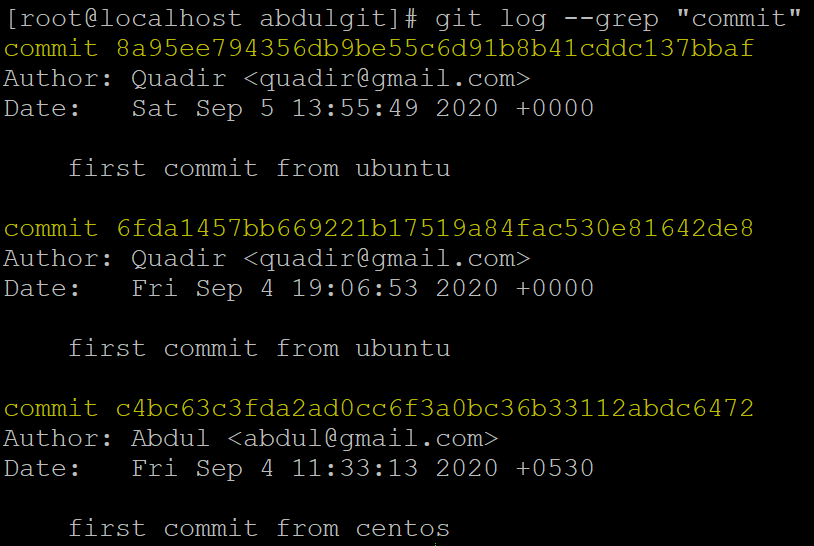
* Git log -2 (to show last 2 commit)



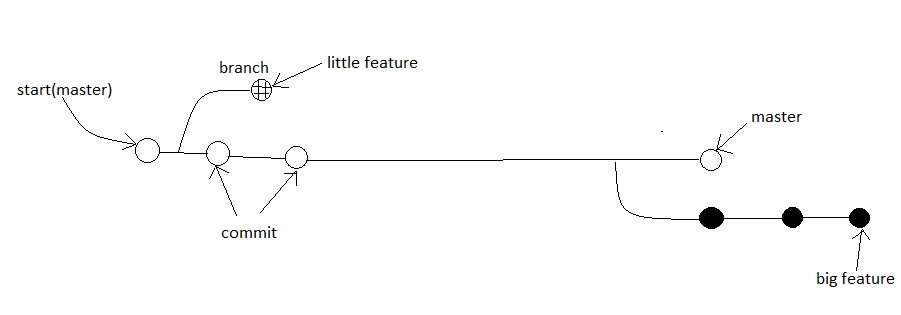
* git log –oneline (to display all commits in one line).



* git log –grep “commit” ( it’s used to search a commit by a word).



**How to create Branch?**

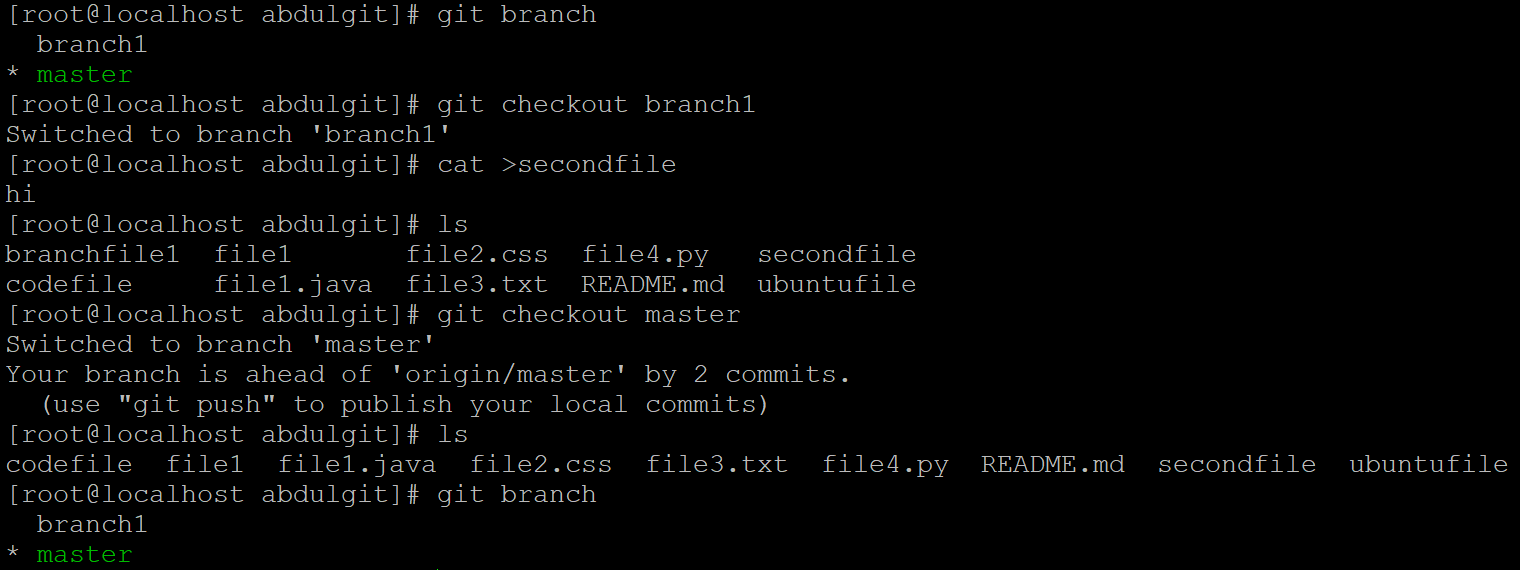


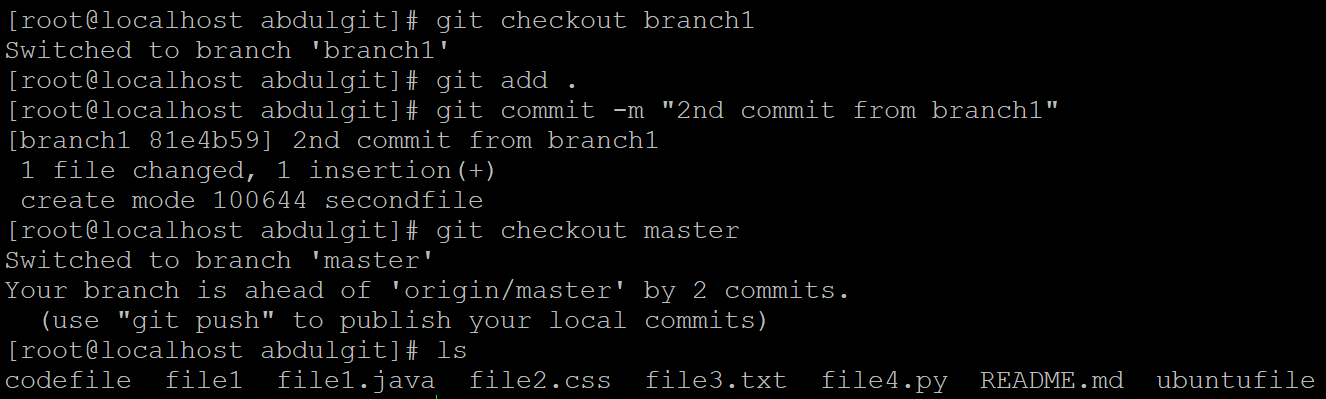
Branch structure

The above diagram visualizes a repository with two isolated lines of development. One for a little feature, by developing them is branches. It’s not possible to work on both of them in parallel but it also keeps the main master branch free from error.

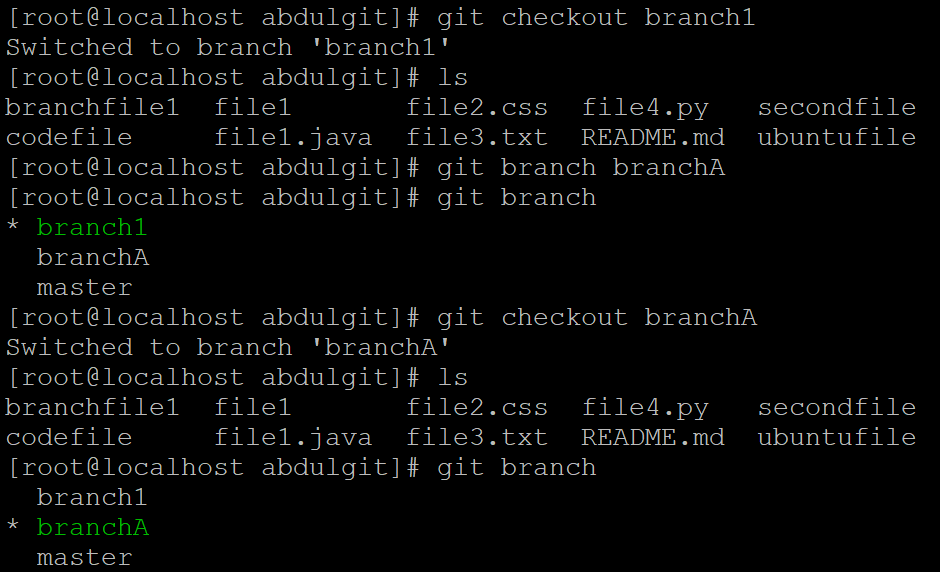
**Properties of Branches**:-

* Each task has one separate branch.
* After done with coding, merge other branches with master.
* This concept is useful for parallel development.
* You can create any number of branches.
* Changes are personal to that particular branch .
* Default branch is ‘master’.
* File created in workspace will be visible in any of the branch workspace until you commit once you commit, then that file belongs to that particular branch.



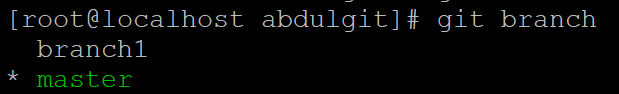


* When created new branch, data of existing branch is copied to new branch.

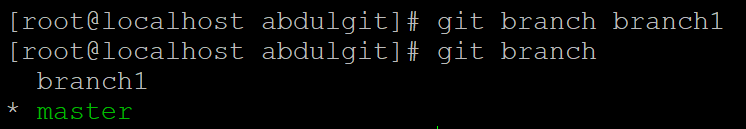


**Commands of Branch:-**

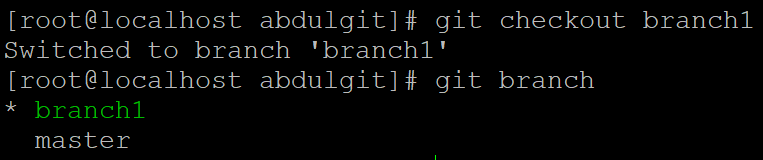
* To see list of available branches (git branch).



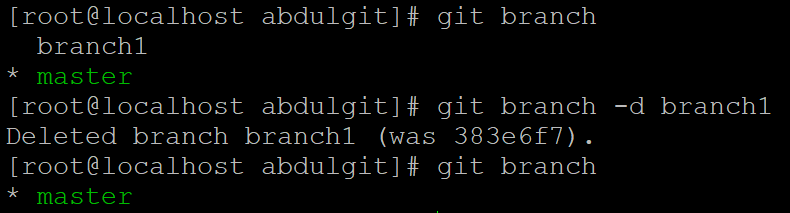
* To create new branch (git branch <branch name>).



* To switch branch. (git checkout <name of switching branch>). (\*) shows your current branch.



* To delete branch (git branch –d/-D <branch name>). [-D is used to delete forcefully].

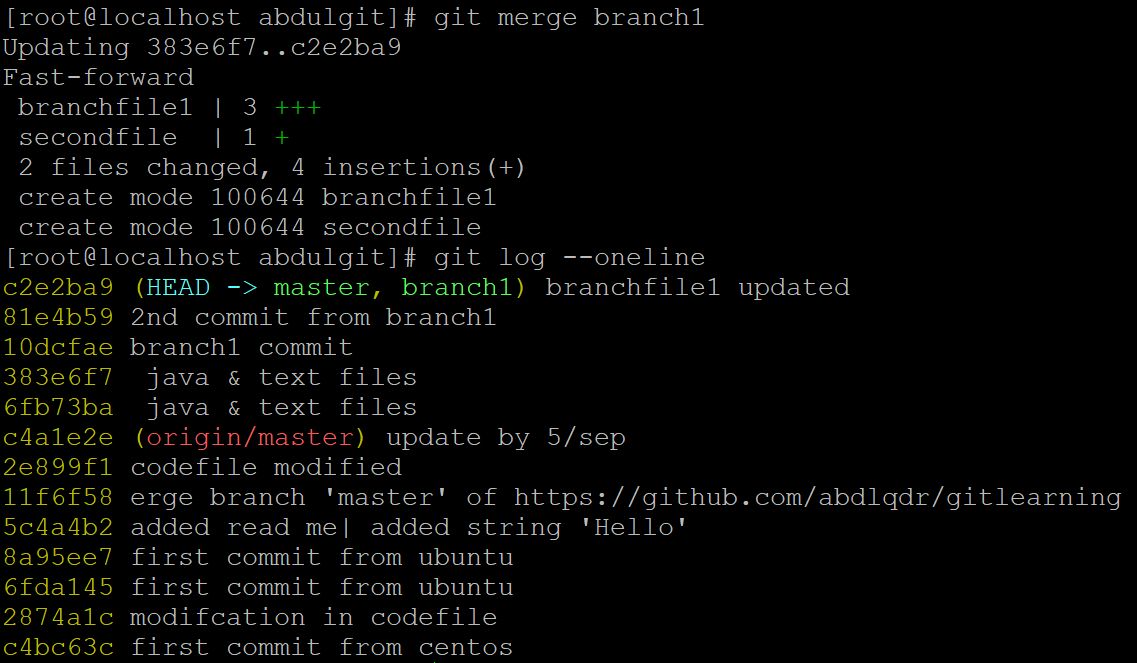


**Branch merge:-**

You can’t merge branches of different repositories.

We use pulling mechanism to merge branches.

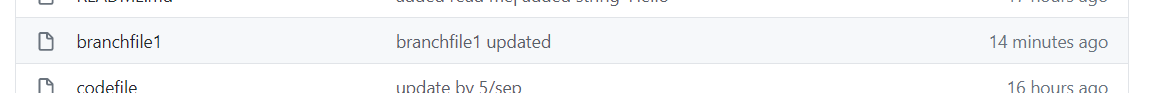
To merge branches (git merge <branch name>). Then run git log –oneline.



Push the commits to github(central repo). (git push origin master)



Check github to see the changes.



**git conflict:-**

* When same file having different content in different branch if you do branch merge, conflict occurs (resolve conflict then add & commit).
* Conflict occurs when you merge branch.

Ex 1:

Branch:Master branch:Branch1

File name:abdl File name:abdl

Content:My name is abdul. Content: My name is abdul.

I live in azamgarh.

* When you merge above branch then there is no conflict because there is same line available in both file. It’ll know second line to merge.

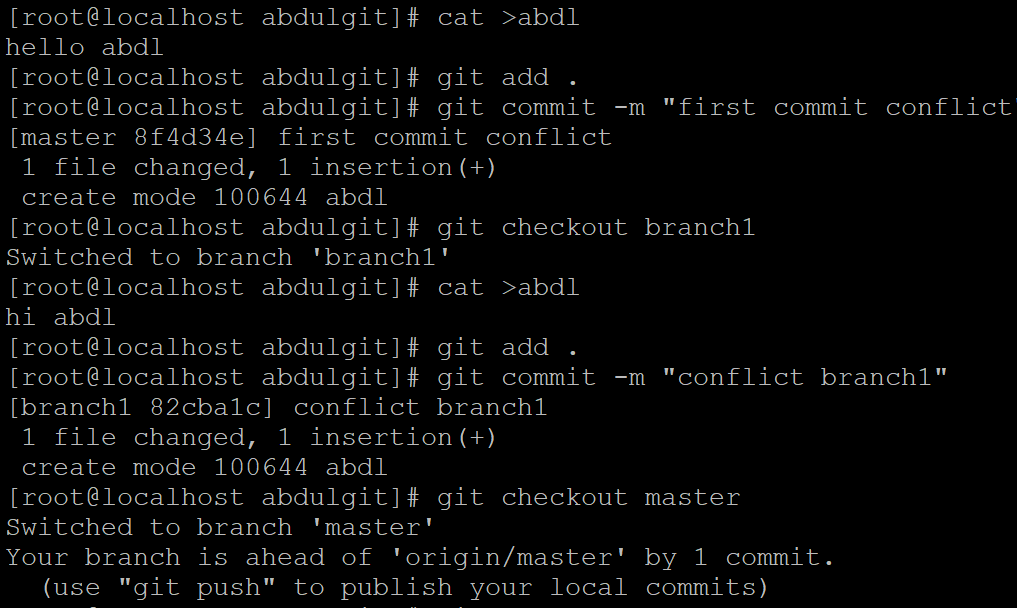
Ex 2:

Branch:Master branch:Branch1

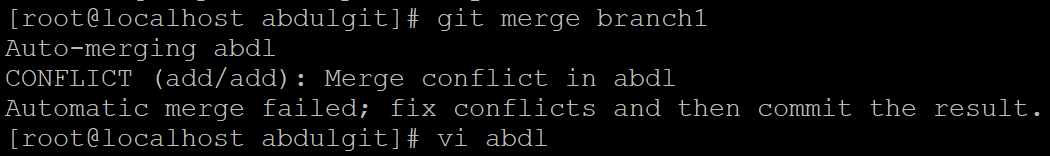
File name:abdl File name:abdl

Content:My name is abdul. Content: I live in azamgarh.

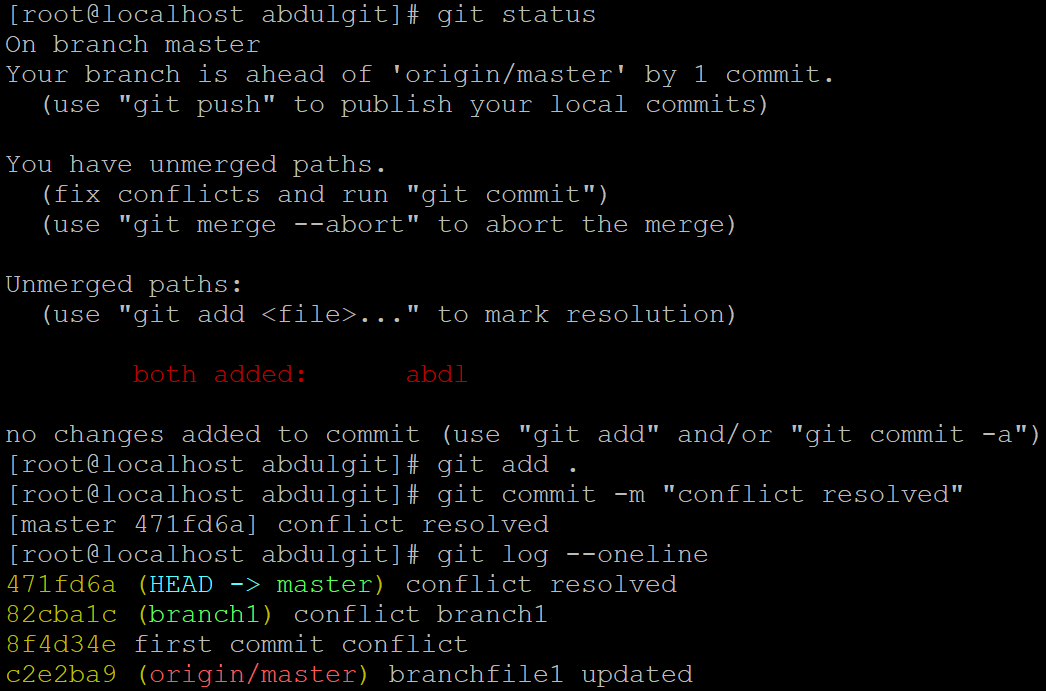
* In above example, when we merge branches git will confuse which data to add above or below. When data is completely different then conflict occurs.



* To correct conflict, In master branch open file in vi editor(vi abdl) & edit as you want & save this file, git understand that & merge file.



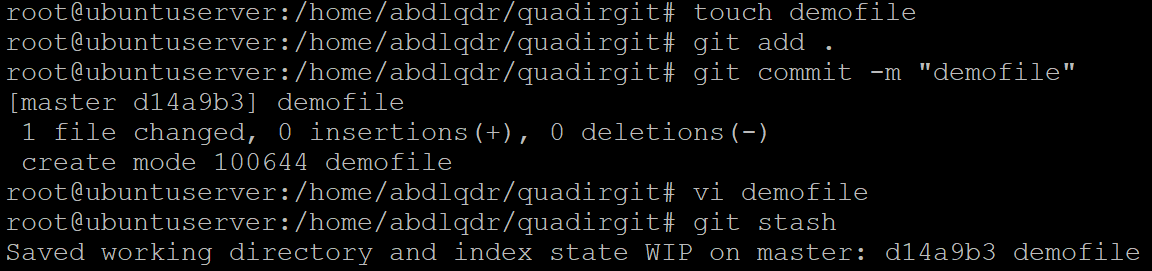
* After that you have to add & commit that merged file in master branch.



**git Stashing:-**

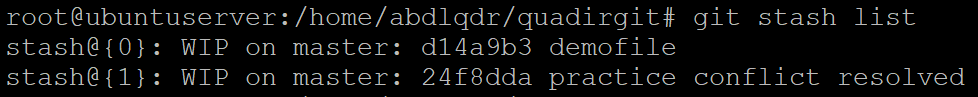
* Suppose, you are implementing a new feature for your product. Your code is in progress & suddenly a customer escalation comes because of this, you have to keep aside your new feature work for few hours. You can’t commit your partial code & also can’t throw away your changes. So, you need some temporary storage, when you can store your partial changes & later on commit it.
* To stash an item (only applies to modified files not new files).

git stash



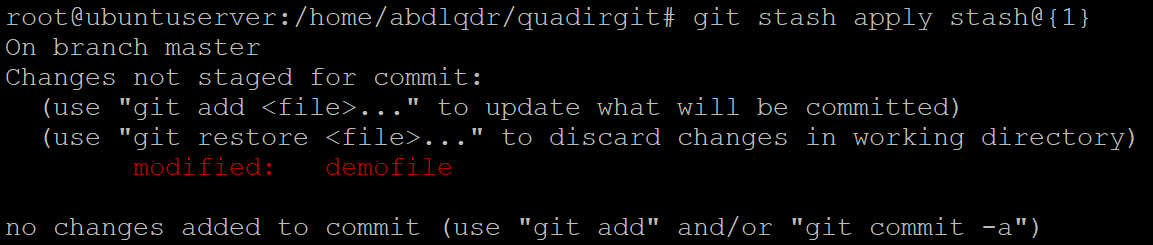
* To see stashed item list

git stash list

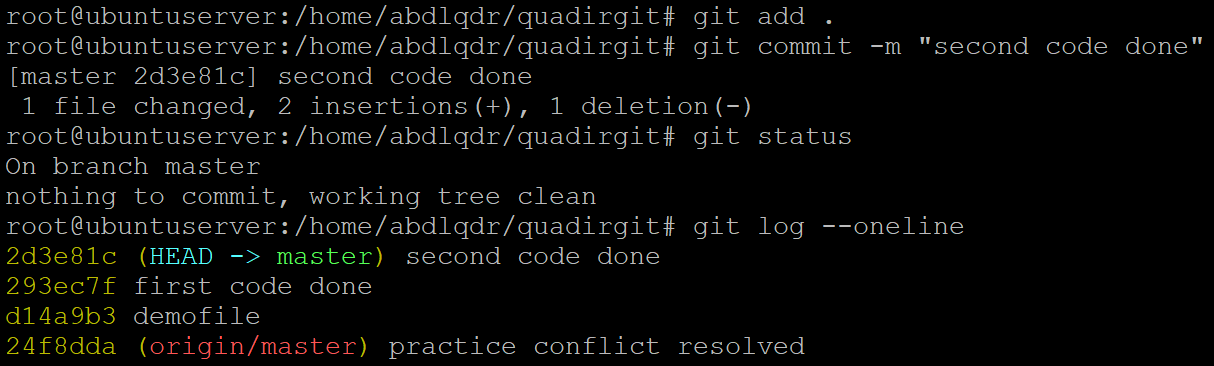


* To apply stashed items(working file will move to workspace).

git stash apply stash stash@{1}

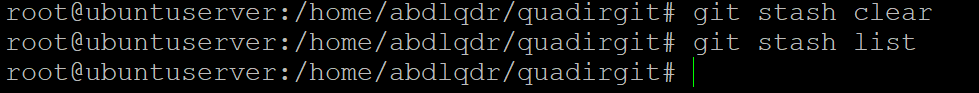


* Then you can add & commit



* To clear the stash items

git stash clear



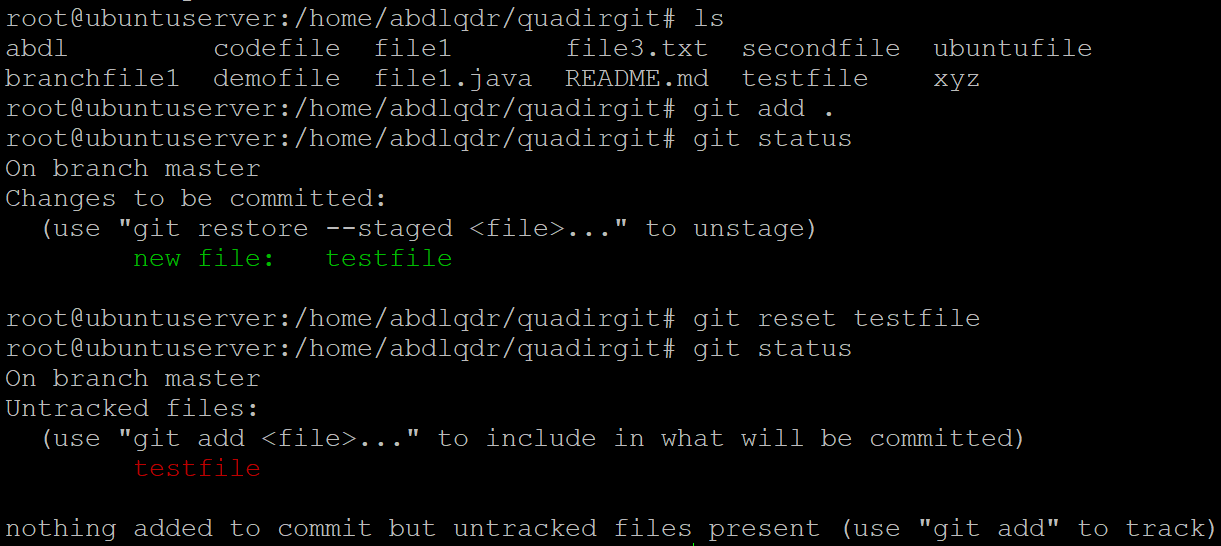
* When you get back your data from stash to workspace. Then that data will be also there in stashing area.

**git reset:-**

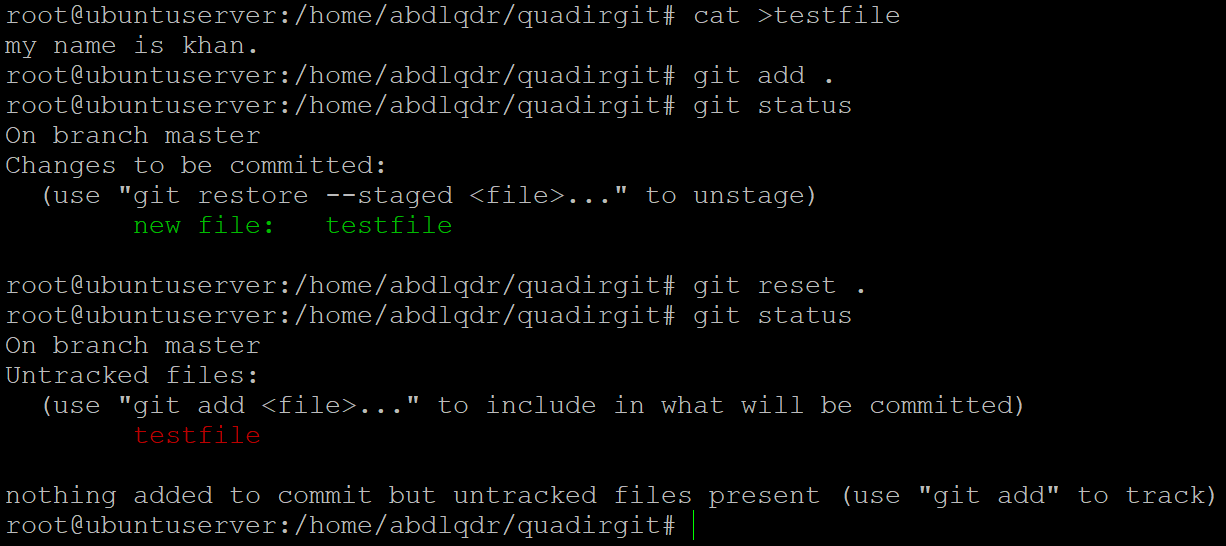
git reset is a powerful command undo local changes to the state of a git repository.

To reset staging area.

git reset <file name>

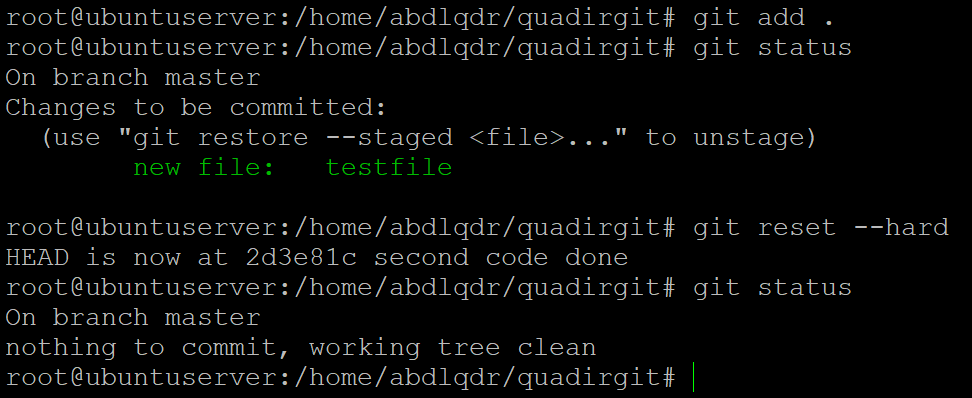


git reset .



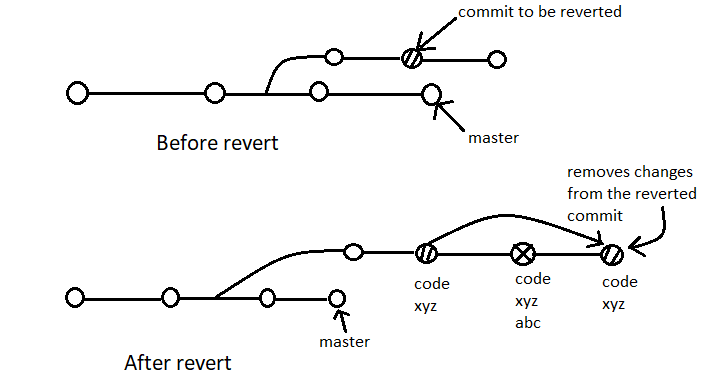
To reset the changes from both staging area & working directory at a time.

git reset –hard



**git revert:-**

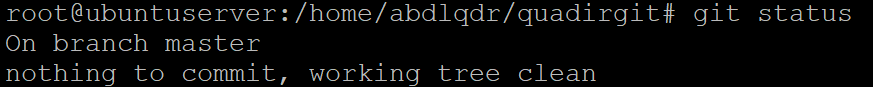
* Revert command helps you undo an existing commit.
* It doesn’t delete any data in the process instead rather git creates new commit with the included files reverted to their previous state. So, your version control history moves forward while the state of your file moves backward.



* When you revert a commit, a commit id is assigned to reverted commit.

**Command for git revert**:-

git status

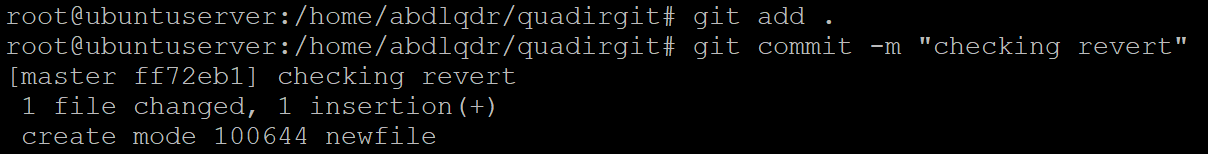


cat >newfile

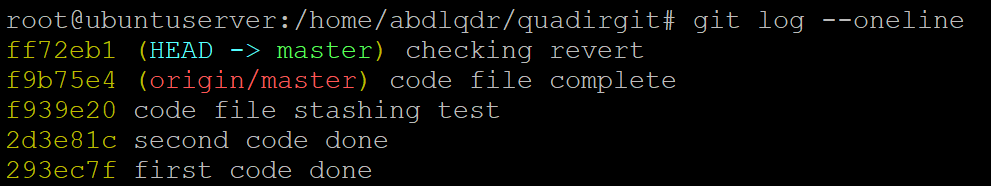


git add .

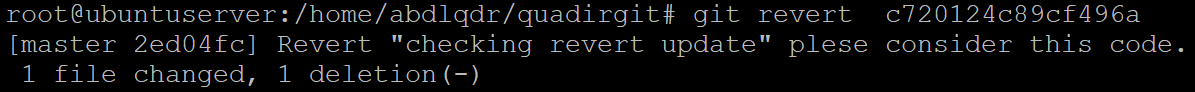
git commit –m “code”



git log –oneline



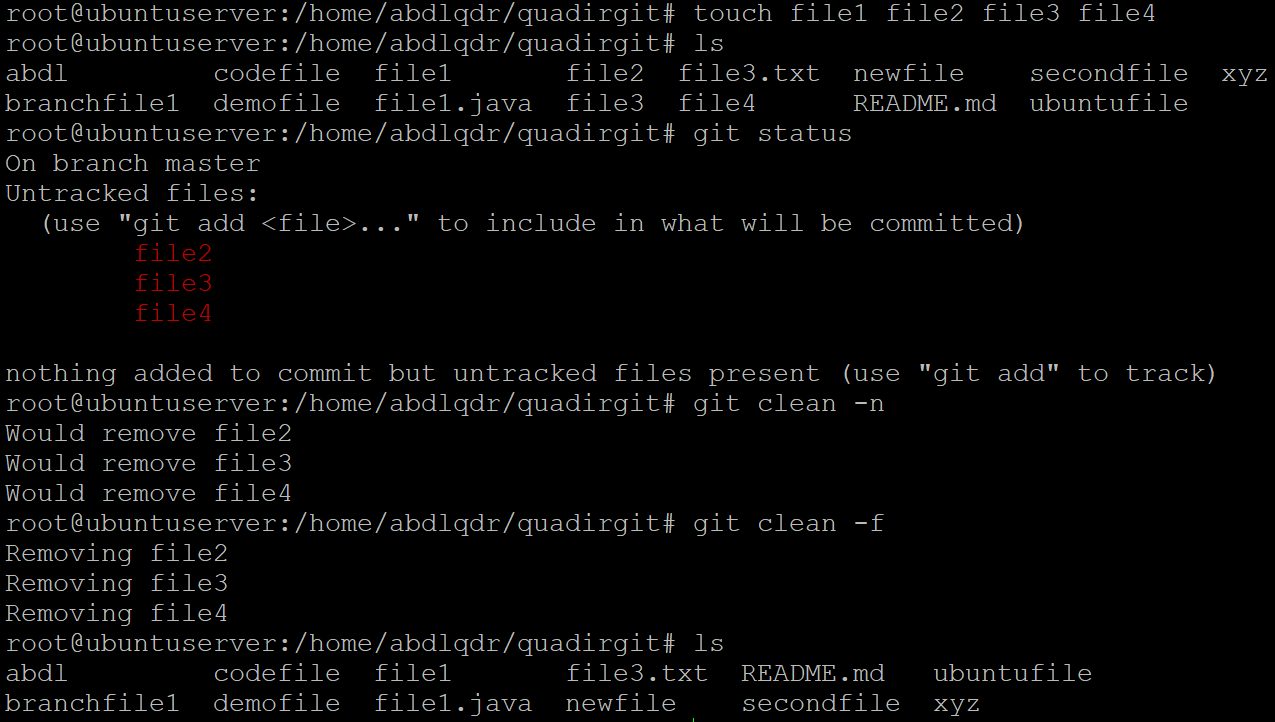
git revert <commit id>



**How to remove untracked files?**

git clean –n (dry run)

git clean –f (forcefully)



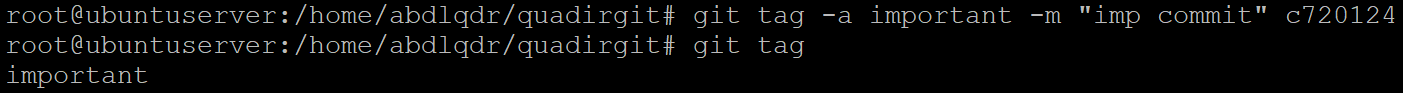
**Tag in git:-**

Tag operation allows you giving meaningful names to a specific version in the repository.

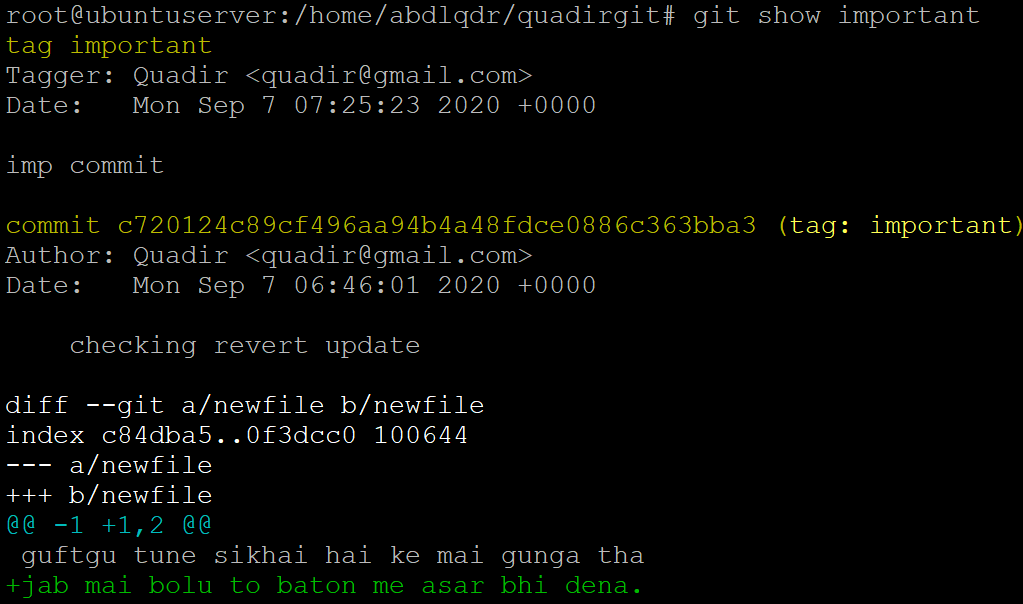
To apply tag.

git tag –a <tag name> -m <message> <commit id>

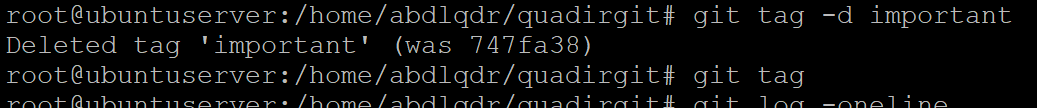
git tag (to see tags)



git show <tag name> (to see particular commit content by using tag)

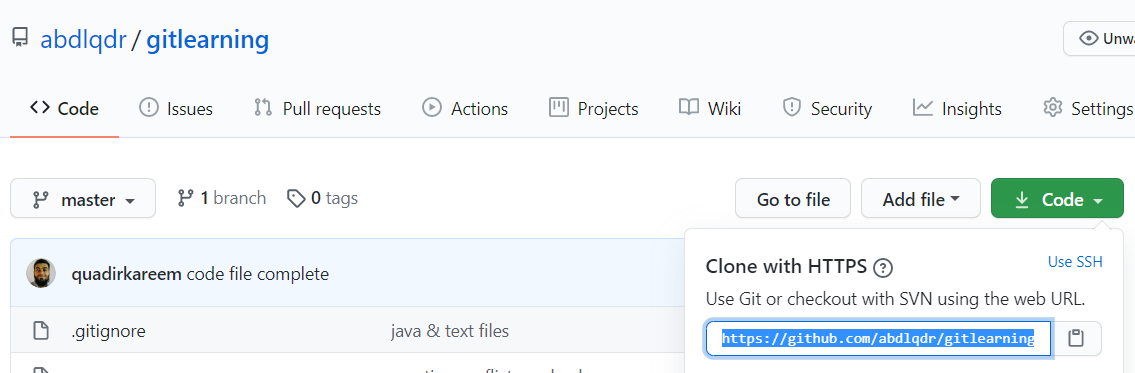


git tag –d <tag name> (to delete tag)



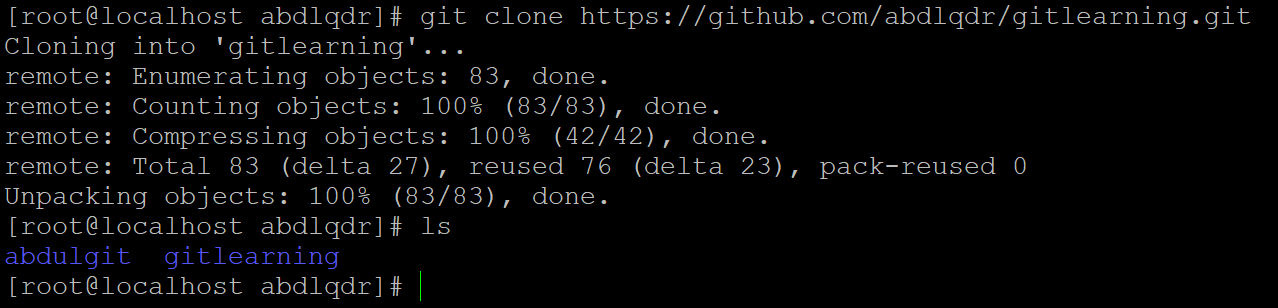
**Github Clone:-**

* Open github website.
* Login & choose existing repository, copy github url.

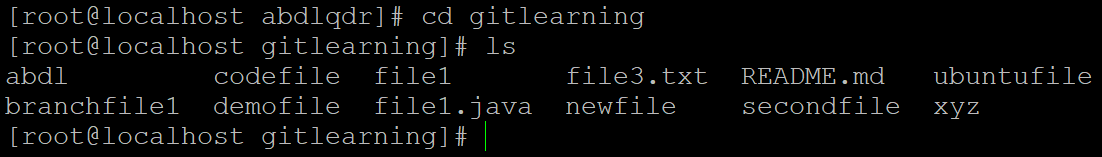


* Now, go to your linux machine, & run command

git clone <url of github repo> (as you can see gitlearning repo folder)



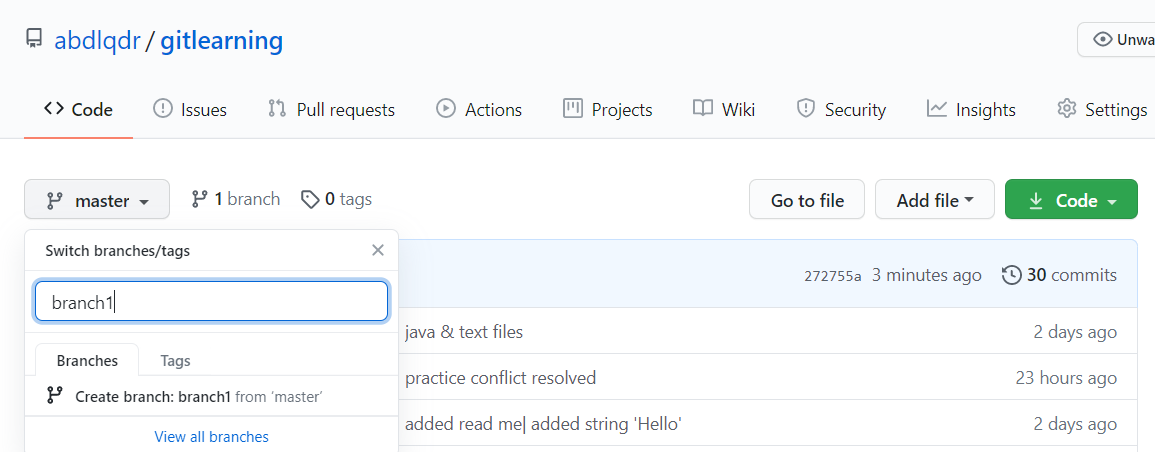
* It creates a local repo automatically in linux machine with the same name as in github account. As you can see above.



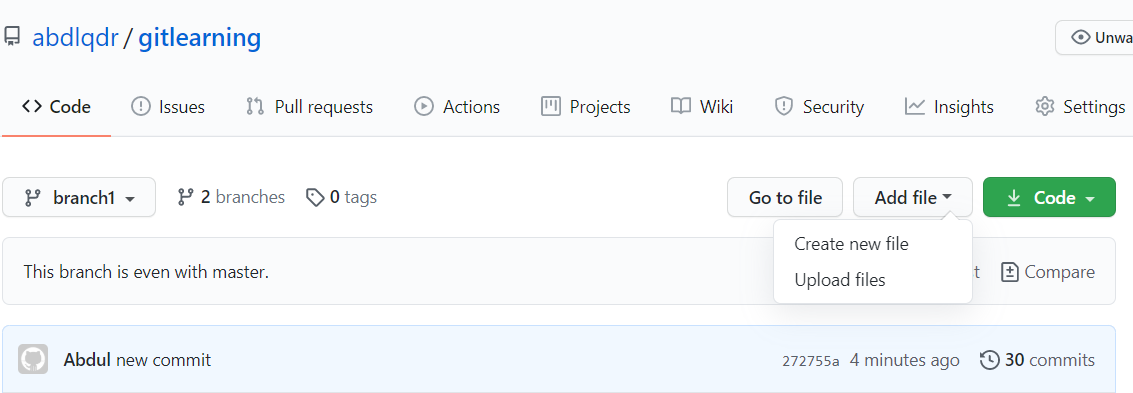
**Github Features:-**

You can make branches in github exactly as you make in git. It’s nature & process is also same as git, like master branch, creation of branches from master branch.

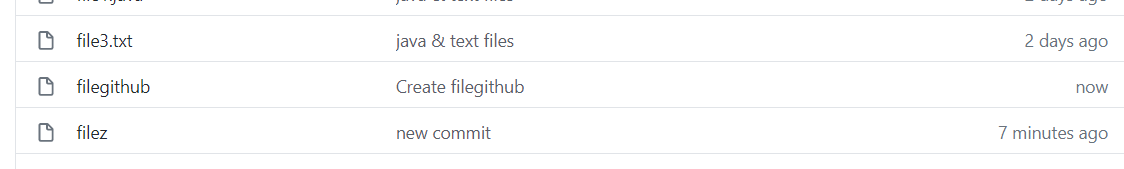
Pulling request is used to see conflict of files when you merge a branch.



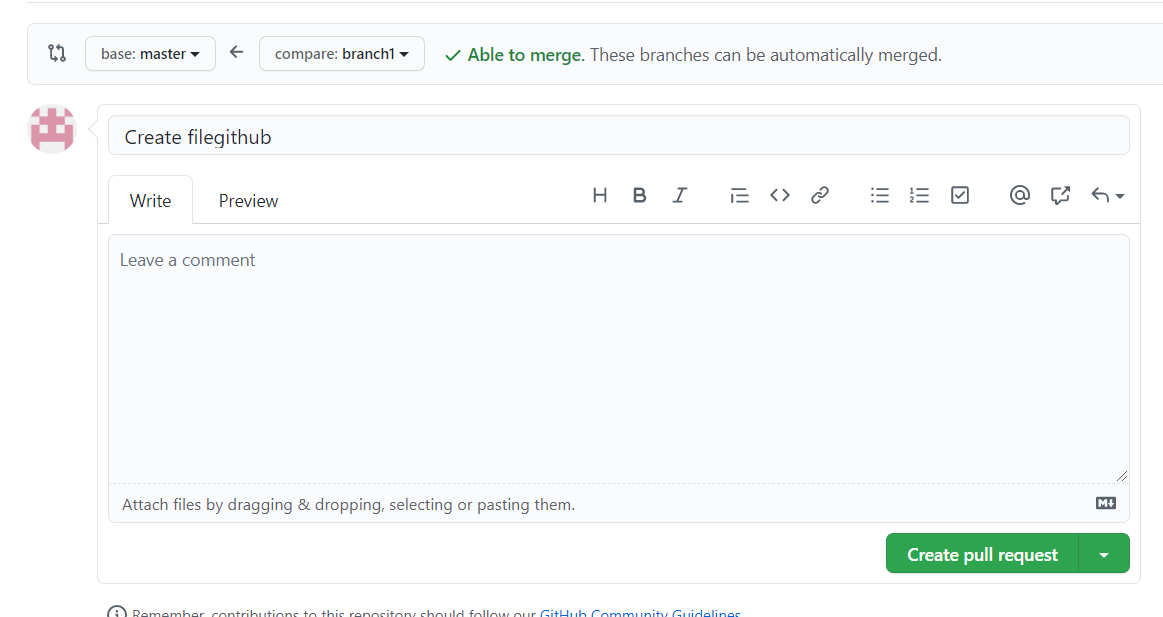
To create file in branch1



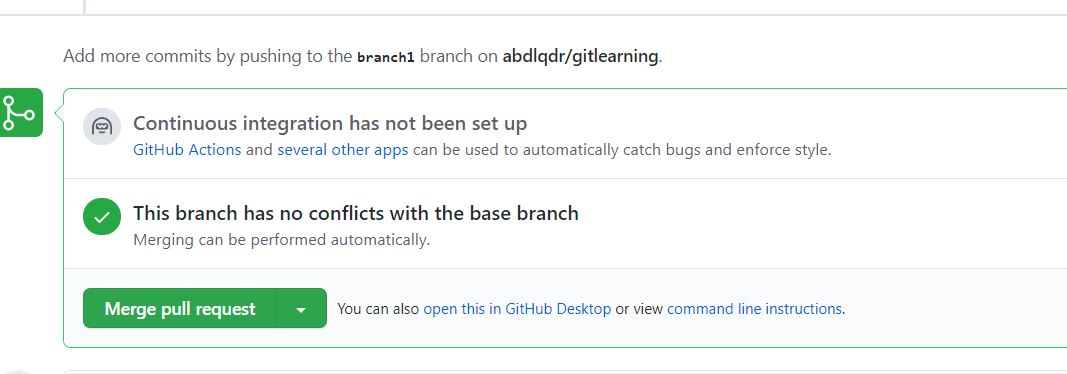
Committed file by github to github.



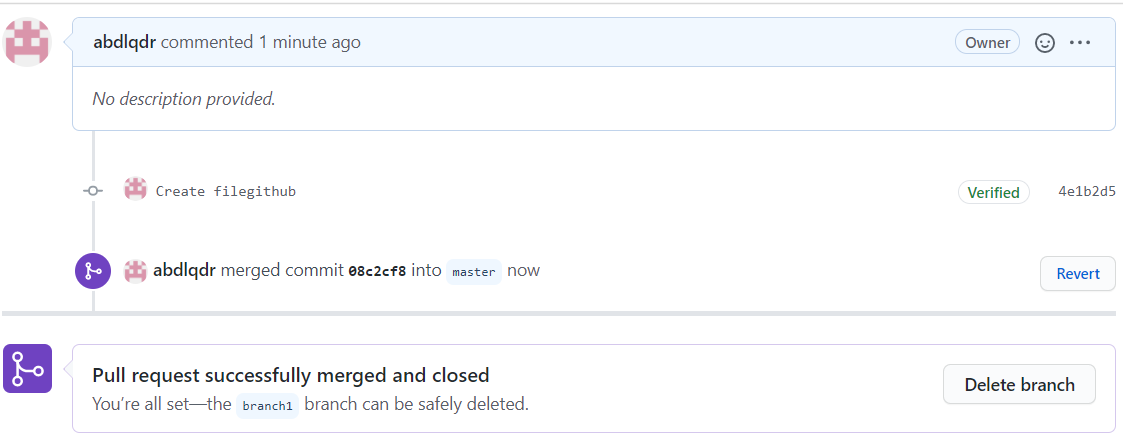
Merging branch1



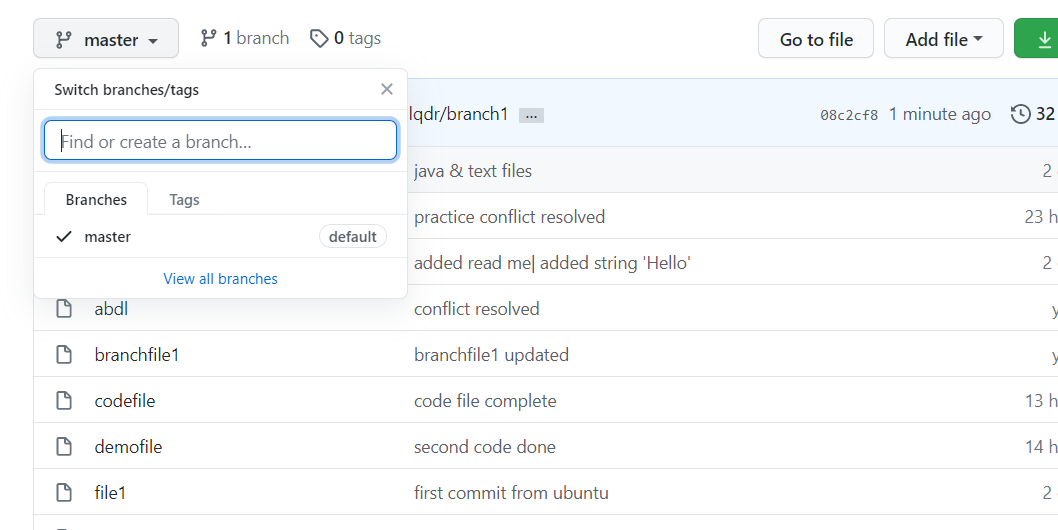
To merge branch.



Click delete branch to delete branch



Only master branch is showing



To delete total repository go to settings & scroll down & click delete repository.

