GIT IT ?

Posted on **May 15, 2020** by **Hridyesh singh bisht**

Do you get git ? , if not then this might be really helpful for you .

Q1.Why know version control ?

It's super important to have detailed historical information for your organization's configuration files and automation code. This let's the administrators see what was modified and when, which can be critical to troubleshooting. It also provides a documentation trail that will let future IT specialists know why the infrastructure is the way it is, and it provides a mechanism for undoing a change completely. This way, we don't have to undo changes from memory and there's less chance of human error.

example :If you are updating your source code for python3 for all machines, and you push this change. Now there are bugs in machines possessing python2, well you could either fix the bugs quickly, but a quick bug fix might not be a permanent solution. Hence, you could do a rollback the the previous source code which worked on all machines and then modify and test your new code, and then update the source code.

Q2.Why keep Historical changes in version control ?

These copies let you see what the project was like before, and go back to that version if you end up deciding that the latest changes were wrong.

They also let you see the progress of the changes over time, and maybe even help you understand why a change was made. We say that this is primitive because it's very manual and not very detailed.

1. First, you need to remember to make the copy.
2. Second, you usually make a copy of the whole thing, even if you're only changing one small part.
3. Third, even if you're emailing your changes to your colleagues, it might be hard to figure out at the end who did what, and more importantly, why they did it.

Q3.When did we git it ?

In 2002, the Linux kernel project began using a proprietary DVCS called Bit-keeper.

In 2005, the relationship between the community that developed the Linux kernel and the commercial company that developed Bit-keeper broke down, and the tool's free-of-charge status was revoked. This prompted the Linux development community (and in particular Linus Torvalds, the creator of Linux) to develop their own tool based on some of the lessons they learned while using Bit-keeper. Some of the goals of the new system were as follows:

- Speed
- Simple design
- Strong support for non-linear development (thousands of parallel branches)
- Fully distributed
- Able to handle large projects like the Linux kernel efficiently (speed and data size)

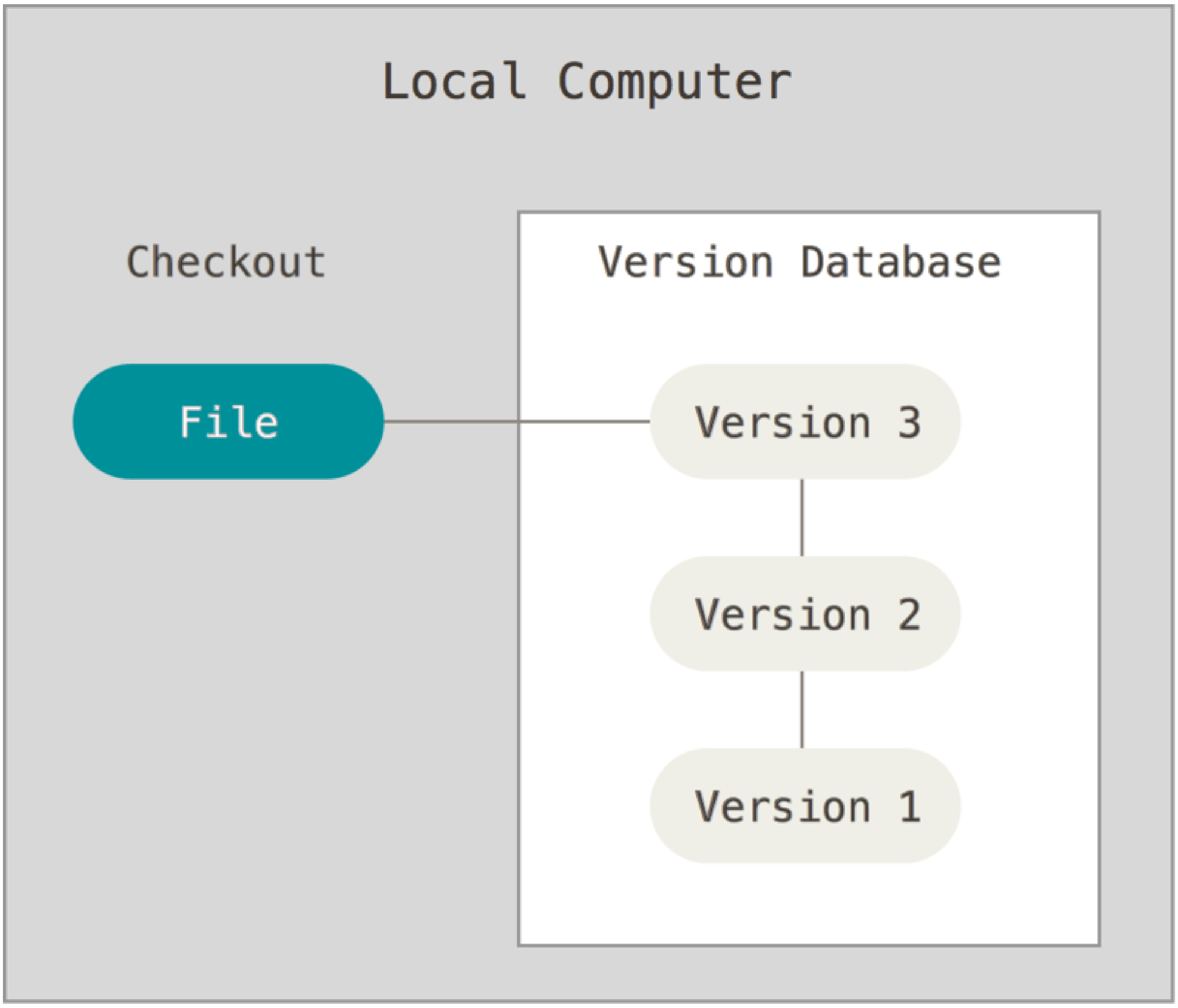
This is the beauty of open source contributions, in my eyes.

Q4.What is version control ?

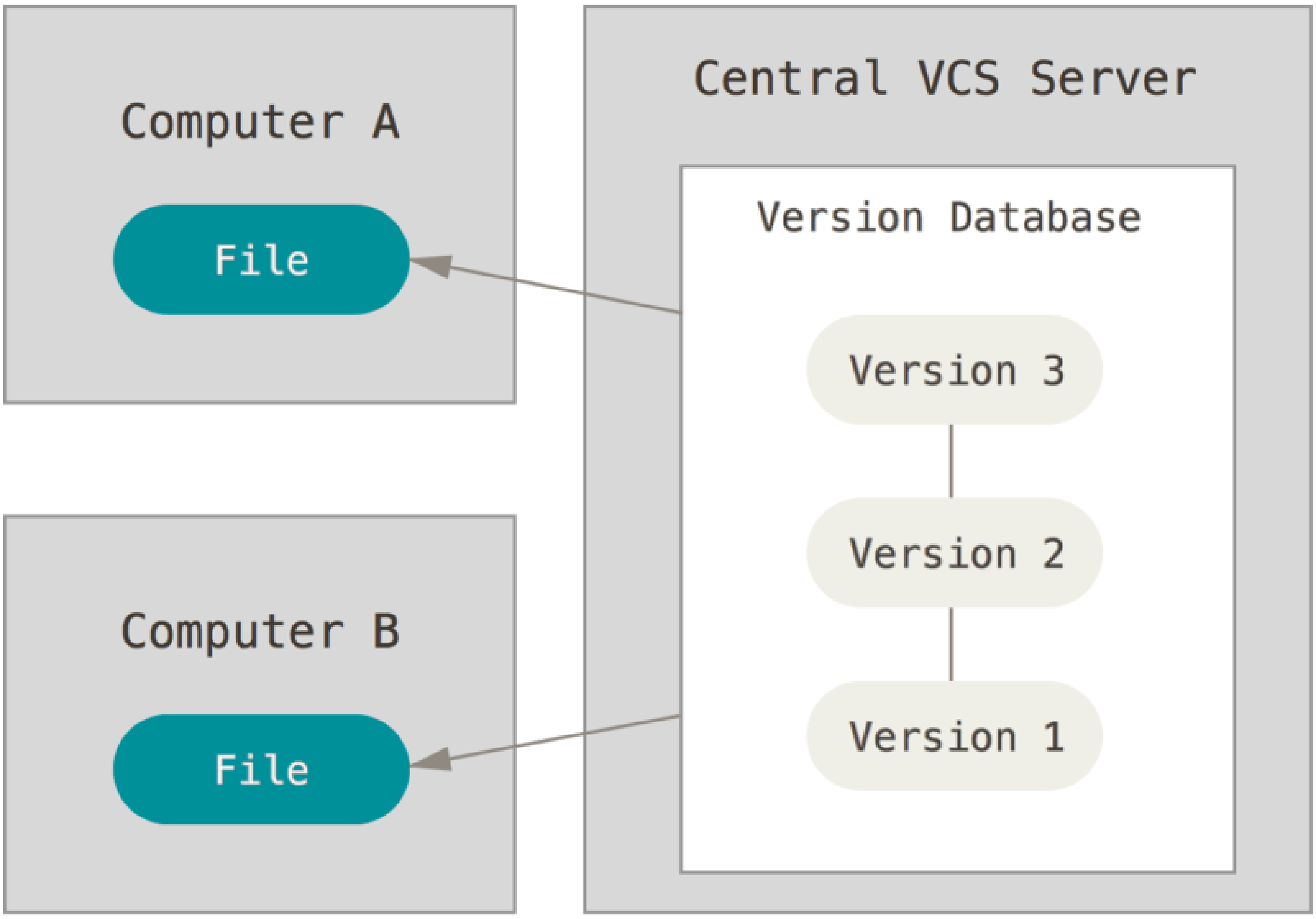
According to Git documents, Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples in a book, you can keep bookmarking at important moments in the book, so you can comeback to that page.

There can be three ways you can utilize it,

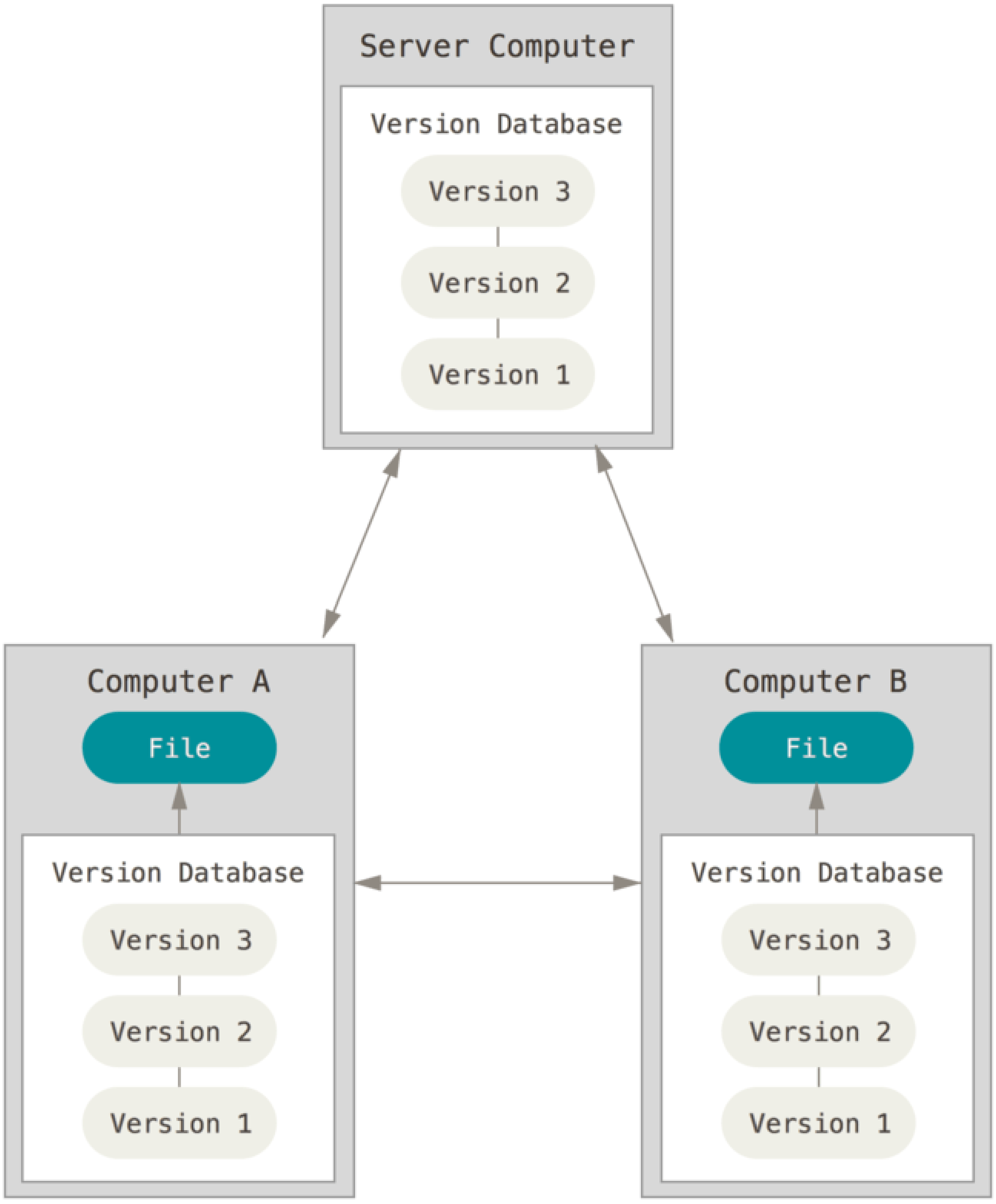
1.Local version, just modifying stuff in your local machine



2.Centralized version, just modifying stuff in your local centralized connection

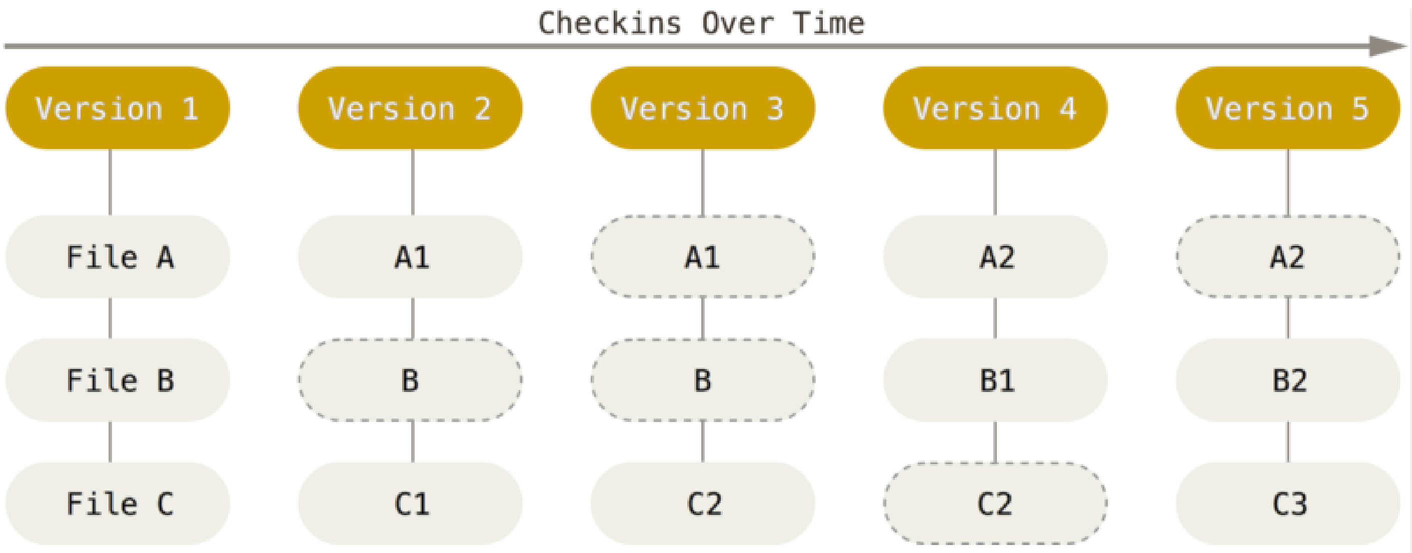


3.Distributed version, this is basically what git is a distributed version control .



Q5.What is GIT?

Git thinks of its data more like a series of snapshots of a miniature filesystem. With Git, every time you commit, or save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. Git thinks about its data more like a **stream of snapshots**.



Commit -> make edits to multiple files and treat that collection of edits as a single change. It also author of the commit to write, what all bugs did he change, why did he do it, etc.

Repository-> contains all related code files, Similar to a directory in Ubuntu.

for more info,

1. **https://git-scm.com/doc (https://git-scm.com/doc)**
2. **https://git-scm.com/docs/git-commit (https://git-scm.com/docs/git-commit)**

1.Installing Git on Ubuntu

1.Command to install git in your PC ,

```
$ sudo apt install git-all
```

2.Check the version of git in your PC , it should be above 2.0

```
$ git --version
```

3.Fill the global name and email settings

```
$ git config --global user.name "Your_Name"
$ git config --global user.email "Your_Email@example.com"
```

4.Select a text editor,

```
$ git config --global core.editor nano
```

5.To check the config files,

```
$ git help config
```

For more info,

1. **https://www.git-scm.com/book/en/v2/Customizing-Git-Git-Configuration (https://www.git-scm.com/book/en/v2/Customizing-Git-Git-Configuration)**

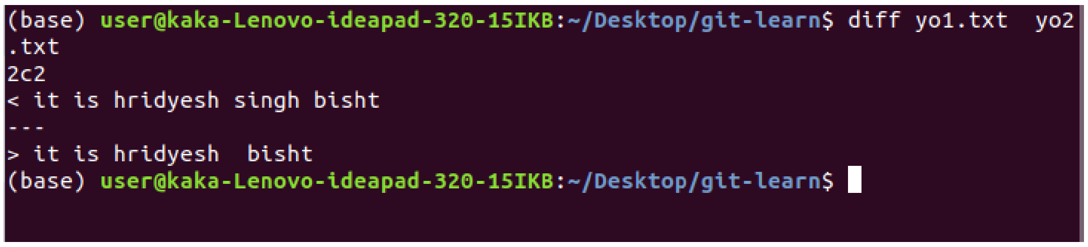
2.commands to help you

Cause now we have documented copy of all the changes happening to the repository , hence we can easy make copies or change the file and integrate it back. How to compare files,

A few terminal file keywords,

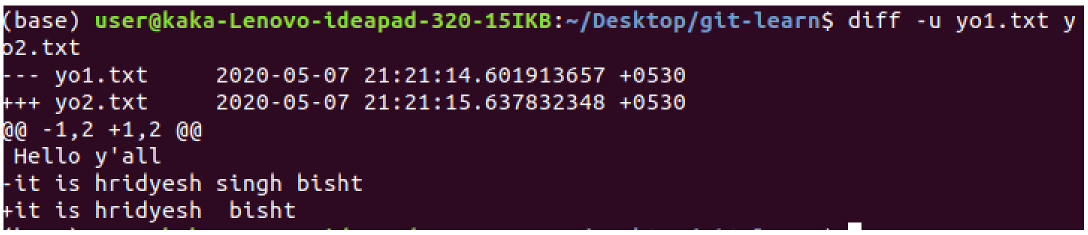
1.diff : tells difference between two files ,

```
diff old_file new_file > change.diff
```



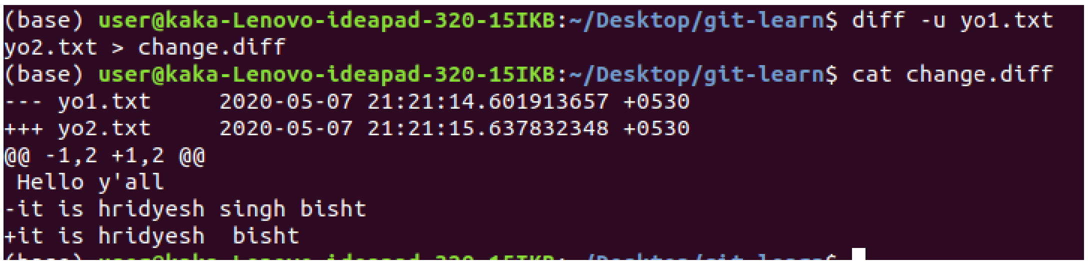
```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn$ diff yo1.txt  yo2
.txt
2c2
< it is hridyesh singh bisht
---
> it is hridyesh  bisht
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn$
```

```
here c -> changed , a -> add, u ->unified format  , -  ->lines removed and + ->
w -diff is used to see the words that have been changed
```



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn$ diff -u yo1.txt y
o2.txt
--- yo1.txt     2020-05-07 21:21:14.601913657 +0530
+++ yo2.txt     2020-05-07 21:21:15.637832348 +0530
@@ -1,2 +1,2 @@
 Hello y'all
-it is hridyesh singh bisht
+it is hridyesh  bisht
```

2.Making a file which contains our changes, done by diff -u old_file new_file > change.diff

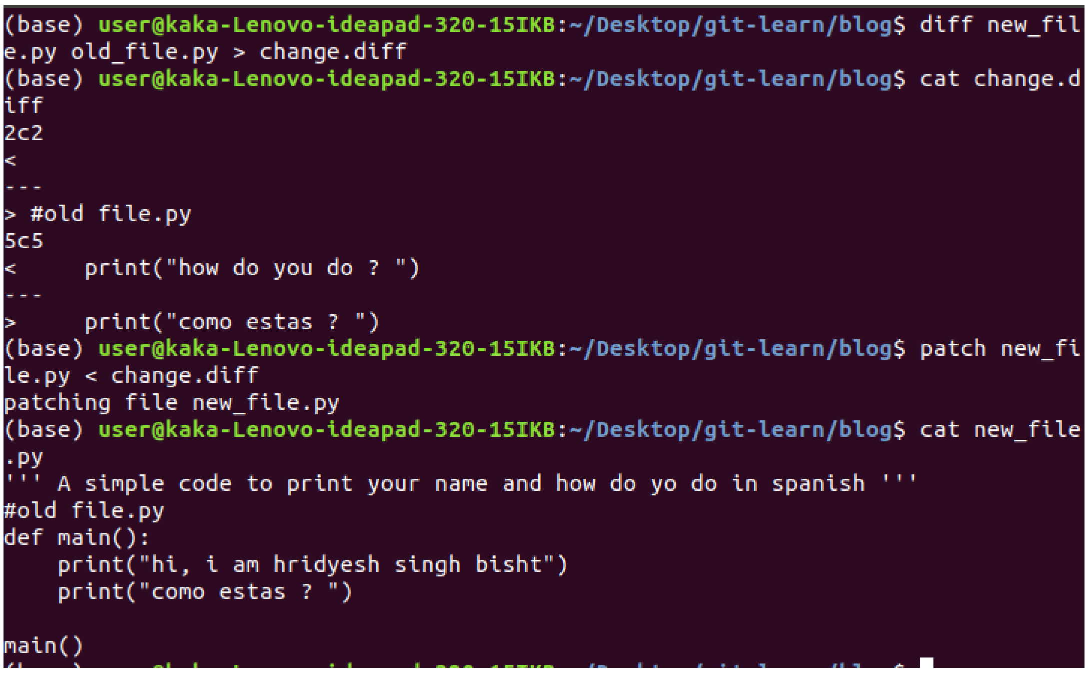


```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn$ diff -u yo1.txt
yo2.txt > change.diff
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn$ cat change.diff
--- yo1.txt     2020-05-07 21:21:14.601913657 +0530
+++ yo2.txt     2020-05-07 21:21:15.637832348 +0530
@@ -1,2 +1,2 @@
 Hello y'all
-it is hridyesh singh bisht
+it is hridyesh  bisht
```

3.patch :takes a file generated by diff and applies it to the original file

```
patch new_file < change.diff
```

< acts as a standard input to a file,



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ diff new_fil
e.py old_file.py > change.diff
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ cat change.d
iff
2c2
<
---
> #old file.py
5c5
<     print("how do you do ? ")
---
>     print("como estas ? ")
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ patch new_fi
le.py < change.diff
patching file new_file.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ cat new_file
.py
''' A simple code to print your name and how do yo do in spanish '''
#old file.py
def main():
    print("hi, i am hridyesh singh bisht")
    print("como estas ? ")

main()
```

4. Practical approach to diff and patch files

Now consider a friend (A) , asked you to fix a code file, now there are two ways to do it.

1. You send me a complete working file from scratch, Now finding the bug in the file you sent will be tough for A.
2. A will send her your file, you will fix the bug and send it back. Which will be easier compared to previous solution but still will be comparatively tough.
3. Using diff and patch,

Now how can you do it using diff and patch,

1. Now A sends a file, then we will copy the code in a new file called improved_file.
2. Now we will work on our improved file, then use diff file to save it into a new file.
3. diff -u A.py improved_file.py < change.diff
4. then send Shruti our change.diff file, so she can see the changes we made.
5. Finally Shruti can patch the change.diff to A.py
6. patch A.py < change.diff.

For more information,

1. **http://man7.org/linux/man-pages/man1/diff.1.html (http://man7.org/linux/man-pages/man1/diff.1.html)**
2. **http://man7.org/linux/man-pages/man1/patch.1.html (http://man7.org/linux/man-pages/man1/patch.1.html)**

6.Basics of GIT

1. First Steps

1. The git directory contains all the changes and their history and the working tree contains the current versions of the files.

2. The staging area which is also known as the index is a file maintained by Git that contains all of the information about what files and changes are going to go into your next command.

Now the steps that happen are,

1. We have to set out name and email in the git as discussed above.
2. We can either initialize a git repository by using git init.
3. Or copy a git repository using git clone URL.
4. After copying the file into git repository, we add it to the staging area
5. git status gives us all the information about working tree
6. git commit, opens a text editor. Then enter the commit message in the text editor.

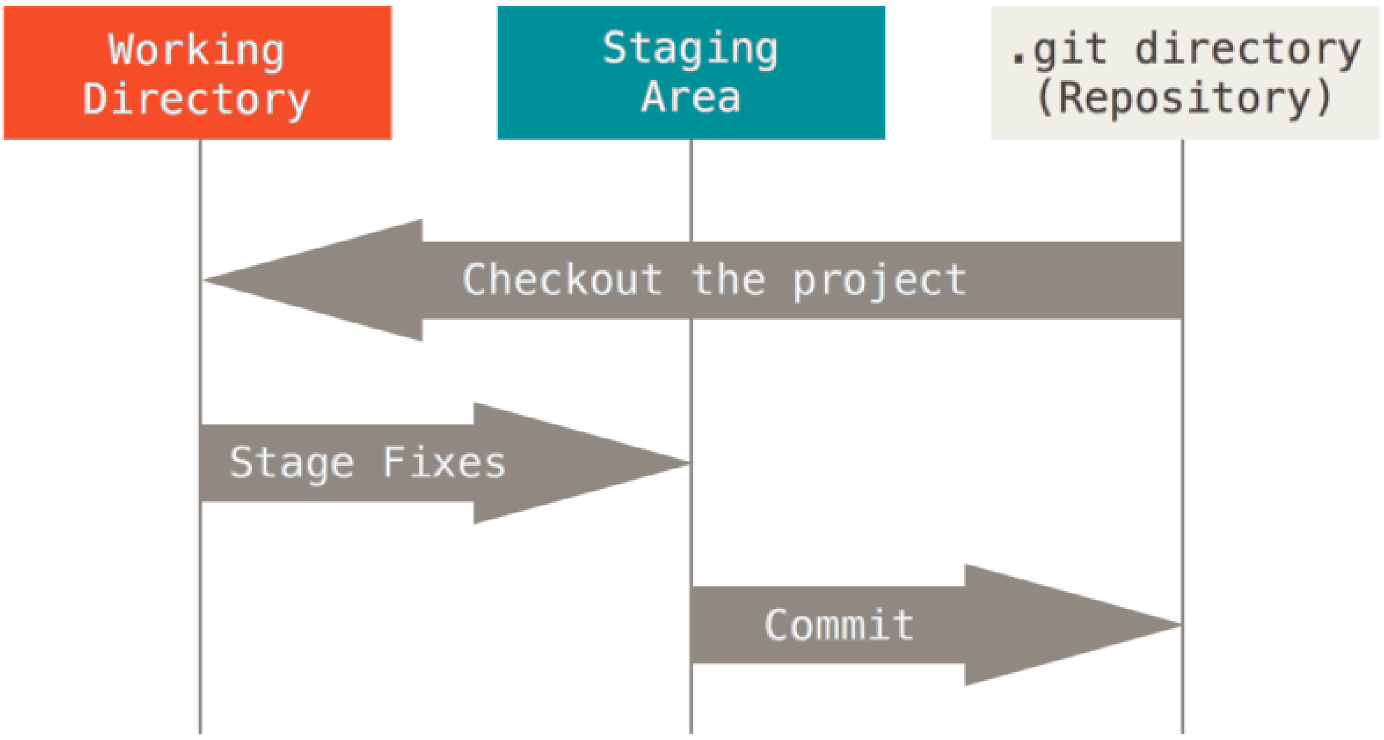
- **If there is no commit message, then it is going to abort the commit message.**

2.Tracking files

1. tracked files : part of the screenshot
2. UN-tracked files : not part of the screenshot

Git has three main states that your files can reside in: *modified*, *staged*, and *committed*:

- Modified means that you have changed the file but have not committed it to your database yet.
- Staged means that you have marked a modified file in its current version to go into your next commit snapshot.
- Committed means that the data is safely stored in your local database.

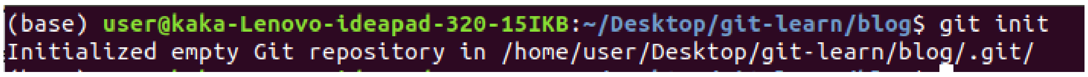


3.Workflow

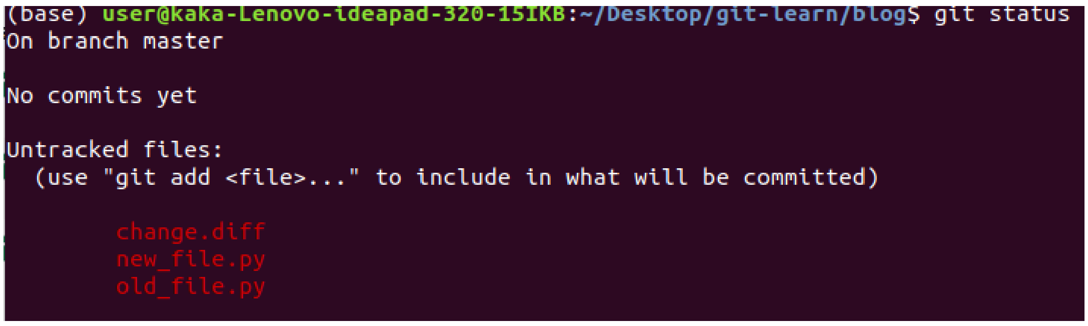
The basic Git workflow goes something like this:

1. You modify files in your working tree.
2. You selectively stage just those changes you want to be part of your next commit, which adds *only* those changes to the staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

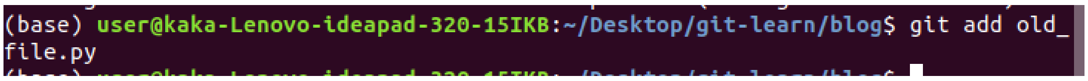
1.Initializing the git Repository


```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git init
Initialized empty Git repository in /home/user/Desktop/git-learn/blog/.git/
```

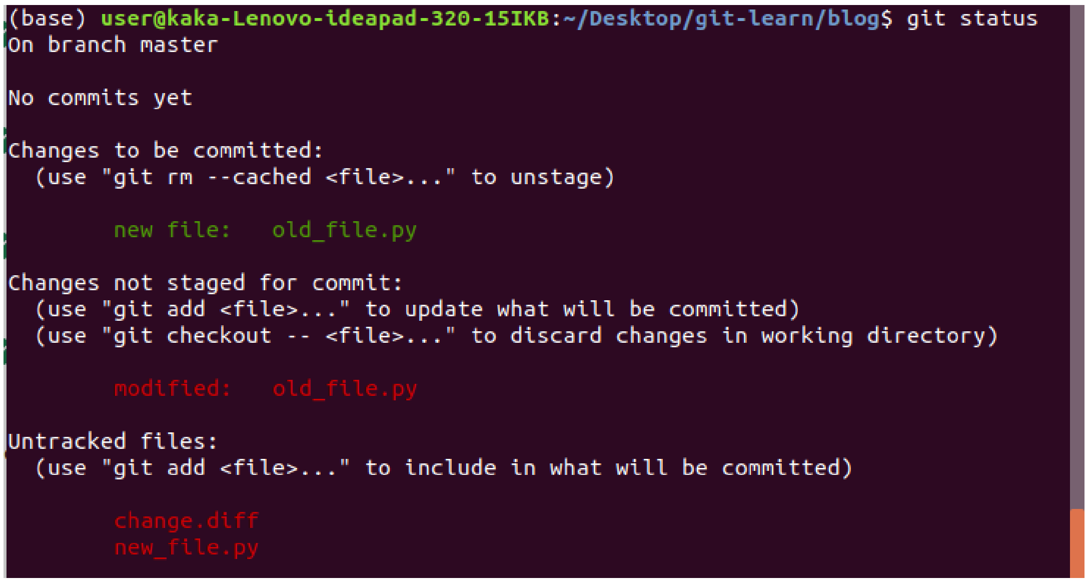
2.git status to show the condition of the file,


```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git status
On branch master

No commits yet

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

        change.diff
        new_file.py
        old_file.py
```

3.git add to add files into the staging area,


```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git add old_
file.py
```

4.git status to show the files added to the staging area,


```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git status
On branch master

No commits yet

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

        new file:   old_file.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   old_file.py

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

        change.diff
        new_file.py
```

5.git push -u origin master : this is the step to push the file to git,

for more info,

1. **https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/process/submitting-patches.rst?id=HEAD (https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/process/submitting-patches.rst?id=HEAD)**
2.

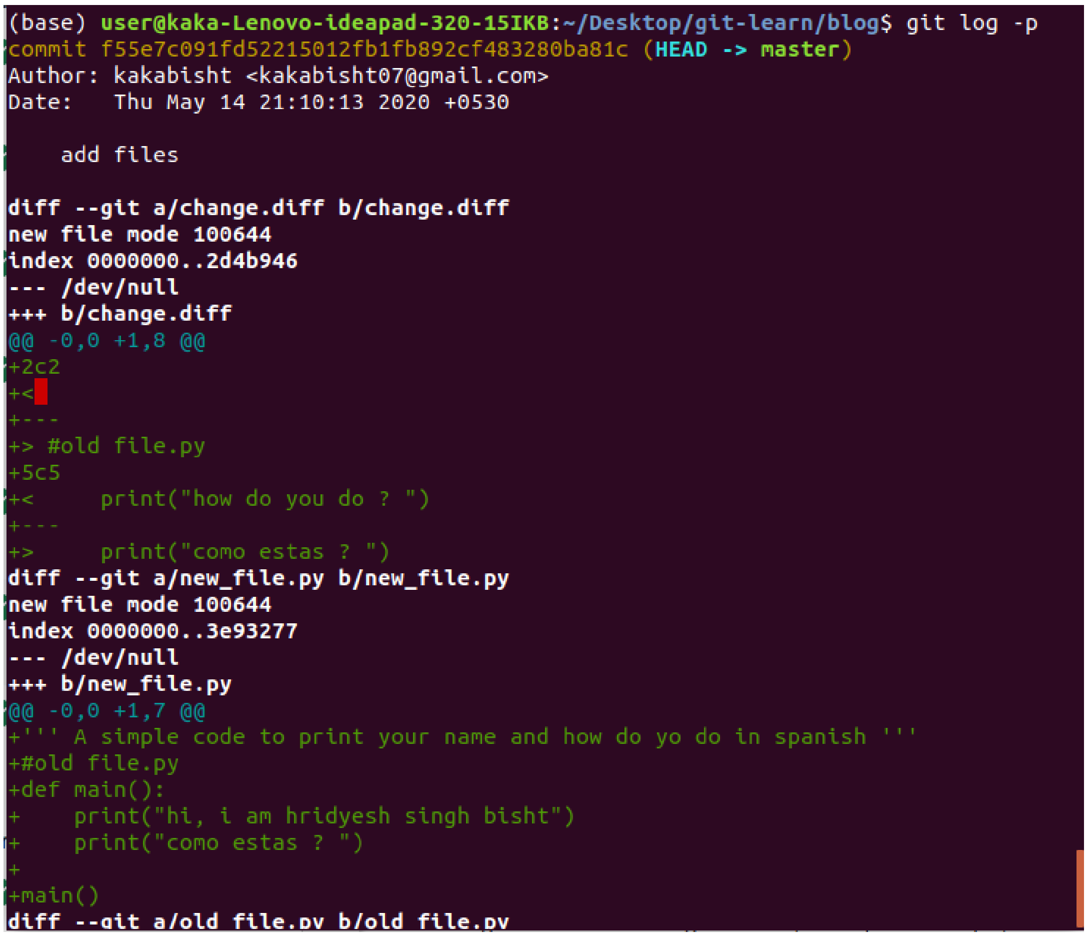
4.Head POINTER IN GIT

1. Git uses the head alias to represent the currently checked out snapshot of your project. This lets you know what the contents of your working directory should be.
2. head is used to indicate what the currently checked out snapshot is
3. We can even use git to go back in time and have head representing old commit from before the latest changes were applied, if a bug comes up.

Example : Think about it as a bookmark that you can use to keep track of where you are. Even if you have multiple books to read, the bookmark allows you to pick up right where you left off.

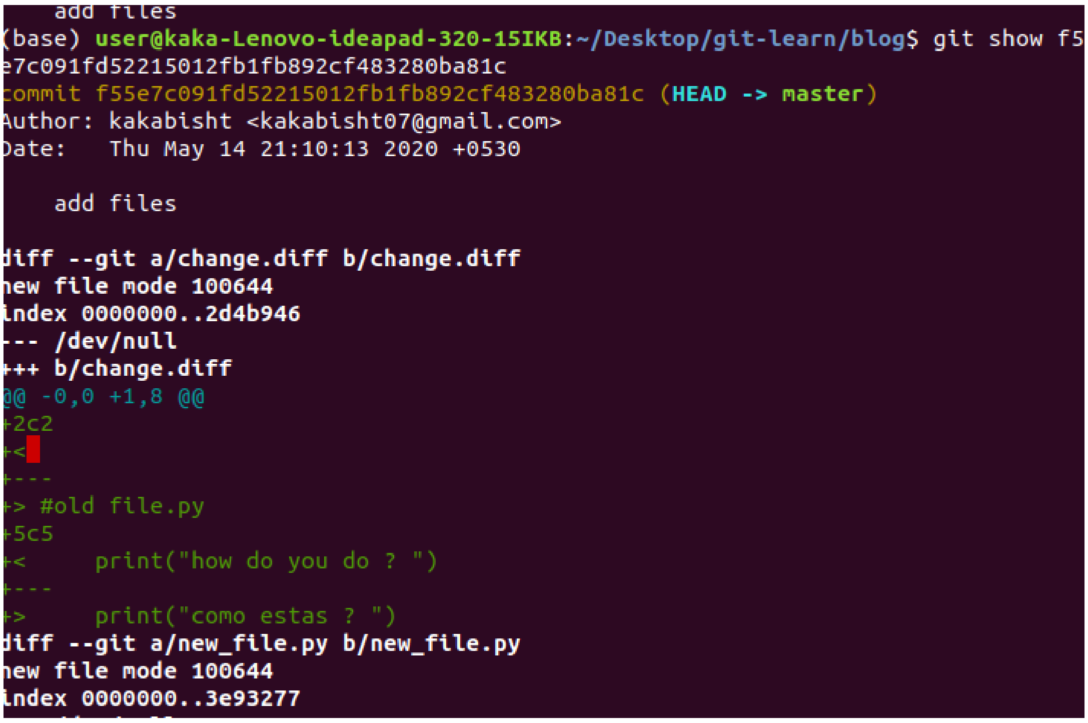
5.More information about our changes

1. git log -p :



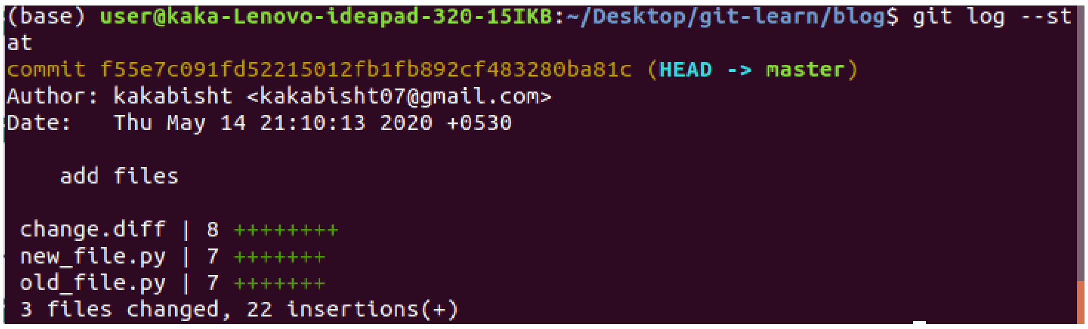
```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git log -p
commit f55e7c091fd52215012fb1fb892cf483280ba81c (HEAD -> master)
Author: kakabisht <kakabisht07@gmail.com>
Date:   Thu May 14 21:10:13 2020 +0530

    add files

diff --git a/change.diff b/change.diff
new file mode 100644
index 0000000..2d4b946
--- /dev/null
+++ b/change.diff
@@ -0,0 +1,8 @@
+2c2
+<
+---
+> #old file.py
+5c5
+<     print("how do you do ? ")
+---
+>     print("como estas ? ")
diff --git a/new_file.py b/new_file.py
new file mode 100644
index 0000000..3e93277
--- /dev/null
+++ b/new_file.py
@@ -0,0 +1,7 @@
+''' A simple code to print your name and how do yo do in spanish '''
+#old file.py
+def main():
+    print("hi, i am hridyesh singh bisht")
+    print("como estas ? ")
+
+main()
diff --git a/old_file.py b/old_file.py
```

2. git show commit-id : to display a particular commit



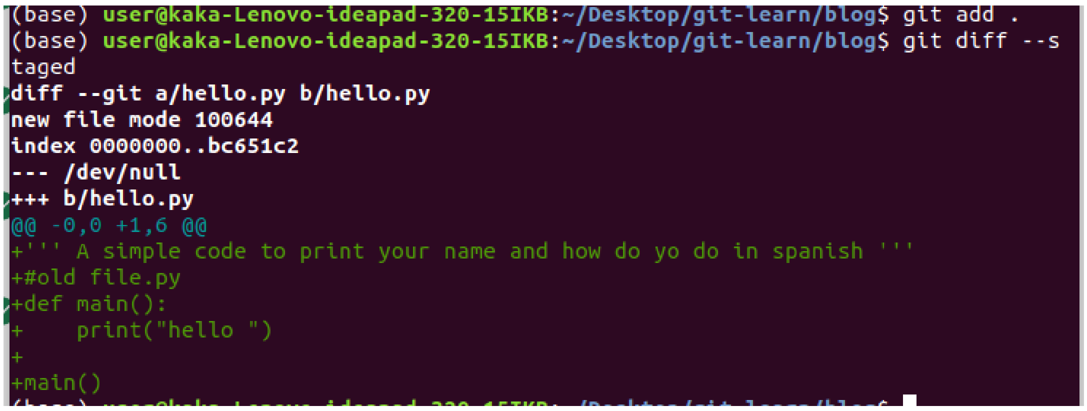
```
    add files
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git show f5
e7c091fd52215012fb1fb892cf483280ba81c
commit f55e7c091fd52215012fb1fb892cf483280ba81c (HEAD -> master)
Author: kakabisht <kakabisht07@gmail.com>
Date:   Thu May 14 21:10:13 2020 +0530

    add files

diff --git a/change.diff b/change.diff
new file mode 100644
index 0000000..2d4b946
--- /dev/null
+++ b/change.diff
@@ -0,0 +1,8 @@
+2c2
+<
+---
+> #old file.py
+5c5
+<     print("how do you do ? ")
+---
+>     print("como estas ? ")
diff --git a/new_file.py b/new_file.py
new file mode 100644
index 0000000..3e93277
```

3.git log — stat



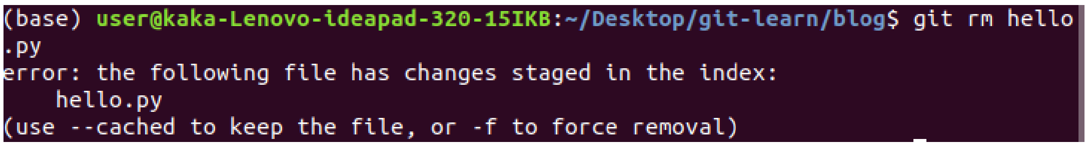
```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git log --st
at
commit f55e7c091fd52215012fb1fb892cf483280ba81c (HEAD -> master)
Author: kakabisht <kakabisht07@gmail.com>
Date:   Thu May 14 21:10:13 2020 +0530

    add files

 change.diff | 8 ++++++++
 new_file.py | 7 +++++++
 old_file.py | 7 +++++++
 3 files changed, 22 insertions(+)
```

4.git diff –staged : staged but no committed



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git add .
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git diff --s
taged
diff --git a/hello.py b/hello.py
new file mode 100644
index 0000000..bc651c2
--- /dev/null
+++ b/hello.py
@@ -0,0 +1,6 @@
+''' A simple code to print your name and how do yo do in spanish '''
+#old file.py
+def main():
+    print("hello ")
+
+main()
```

5.git rm to remove a file



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git rm hello
.py
error: the following file has changes staged in the index:
    hello.py
(use --cached to keep the file, or -f to force removal)
```

.gitignore files

.gitignore files are used to tell the git tool to intentionally ignore some files in a given Git repository. For example, this can be useful for configuration files or metadata files that a user may not want to check into the master branch. Check out more at: **https://git-scm.com/docs/gitignore (https://git-scm.com/docs/gitignore)**.

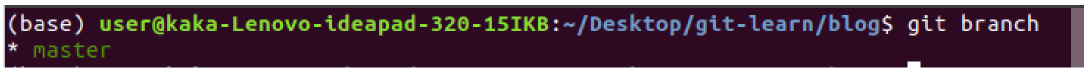
6.Branches

In Git, a branch at the most basic level is just a pointer to a particular commit. But more importantly, it represents an independent line of development in a project. Of which the commit it points to is the latest link in a chain of developing history. The default branch that Git creates for you when a new repository initialized is called master.

master branch is for the working source code, where as branches are used to new features or fixes, that might work. So, a branch helps you try out these features without corrupting the actual source code.

Example : Imagine you have an assignment, you are going to keep few notebooks one for the final submission (master branch), which will be neat and clean, others in which you have rough draft of the submission (branches).

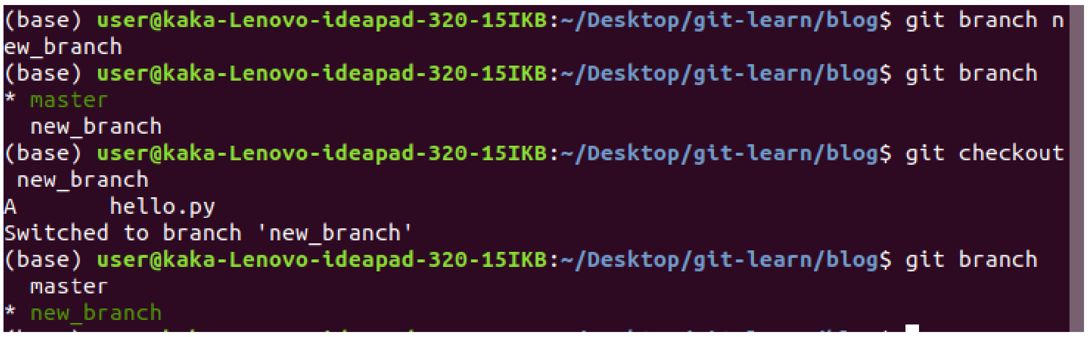
1.git branch : shows all the branches present.



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git branch
* master
```

2.git new branch new_branch_name : to create a new branch

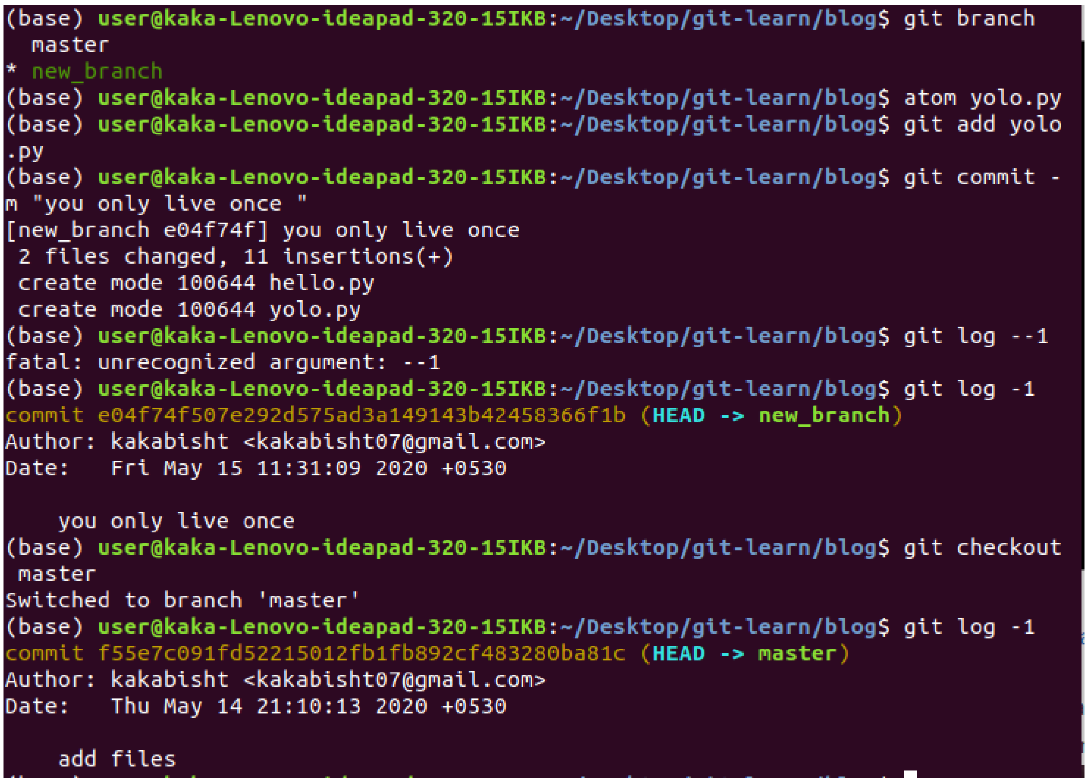
3.git checkout branch_name : to change the branch we are working on



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git branch n
ew_branch
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git branch
* master
  new_branch
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git checkout
 new_branch
A       hello.py
Switched to branch 'new_branch'
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git branch
  master
* new_branch
```

the green (*) shows the current branch we are on.

Now the practical demo, let us consider we are adding a new file in new_branch know as yolo.py, now when we add and commit this file in the new_branch it will only be present in new_branch not in master branch. As you can see, there is a difference in the log files between two branches,

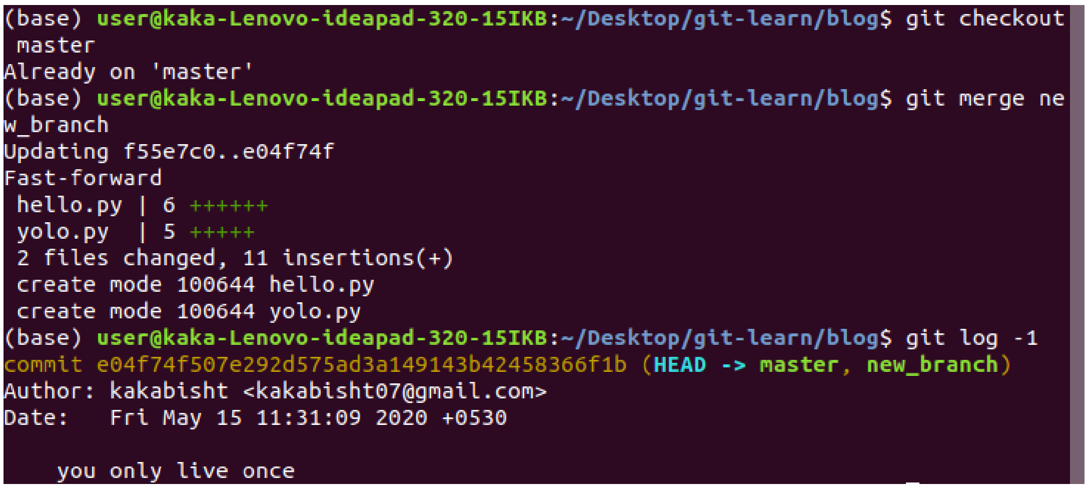


```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git branch
  master
* new_branch
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ atom yolo.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git add yolo
.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git commit -
m "you only live once "
[new_branch e04f74f] you only live once
 2 files changed, 11 insertions(+)
 create mode 100644 hello.py
 create mode 100644 yolo.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git log --1
fatal: unrecognized argument: --1
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git log -1
commit e04f74f507e292d575ad3a149143b42458366f1b (HEAD -> new_branch)
Author: kakabisht <kakabisht07@gmail.com>
Date:   Fri May 15 11:31:09 2020 +0530

    you only live once
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git checkout
 master
Switched to branch 'master'
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git log -1
commit f55e7c091fd52215012fb1fb892cf483280ba81c (HEAD -> master)
Author: kakabisht <kakabisht07@gmail.com>
Date:   Thu May 14 21:10:13 2020 +0530

    add files
```

7.Merging

When our experimental code in new_branch work, we can add that piece of code back to the master branch using merge command.

1. Switch to master branch
2. Merge the new_branch
3. check the log of the master branch

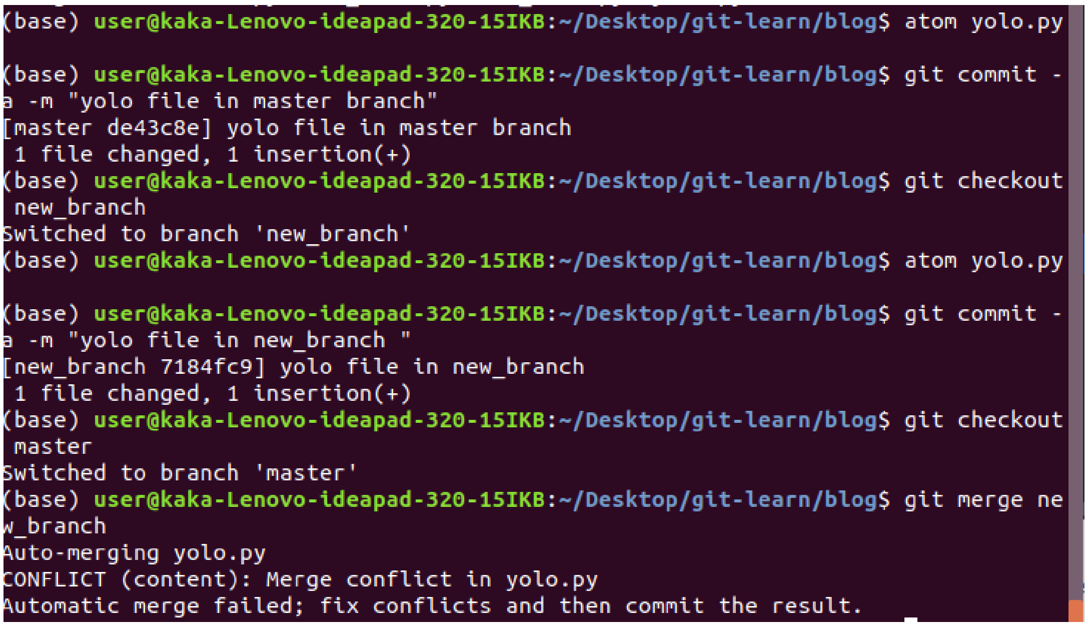


```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git checkout
 master
Already on 'master'
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git merge ne
w_branch
Updating f55e7c0..e04f74f
Fast-forward
 hello.py | 6 ++++++
 yolo.py  | 5 +++++
 2 files changed, 11 insertions(+)
 create mode 100644 hello.py
 create mode 100644 yolo.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git log -1
commit e04f74f507e292d575ad3a149143b42458366f1b (HEAD -> master, new_branch)
Author: kakabisht <kakabisht07@gmail.com>
Date:   Fri May 15 11:31:09 2020 +0530

    you only live once
```

Now head pointer points at both master branch as well as new_branch, there are two main merging algorithms.

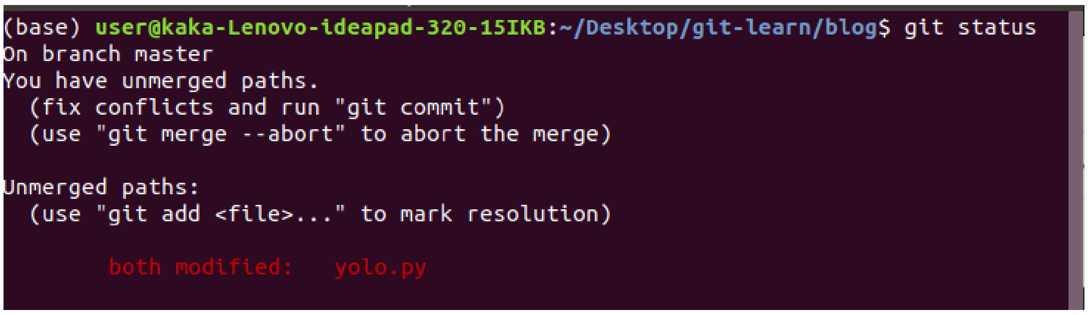
1. fast forward merge : linear path
2. three way merge : non linear path

Now in three way merge, if we change same content of same file on two different branches, then add and commit them. When we merge them git wont know which command to consider or to discard, hence merge conflict arises.



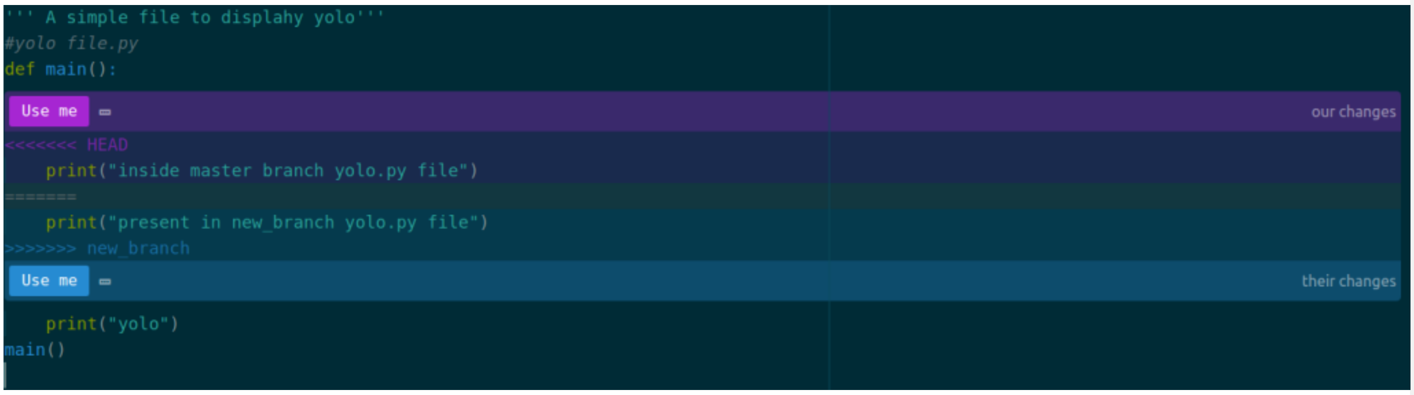
```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ atom yolo.py

(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git commit -
a -m "yolo file in master branch"
[master de43c8e] yolo file in master branch
 1 file changed, 1 insertion(+)
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git checkout
 new_branch
Switched to branch 'new_branch'
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ atom yolo.py

(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git commit -
a -m "yolo file in new_branch "
[new_branch 7184fc9] yolo file in new_branch
 1 file changed, 1 insertion(+)
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git checkout
 master
Switched to branch 'master'
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git merge ne
w_branch
Auto-merging yolo.py
CONFLICT (content): Merge conflict in yolo.py
Automatic merge failed; fix conflicts and then commit the result.
```

now we use git status to help us, what is happening here



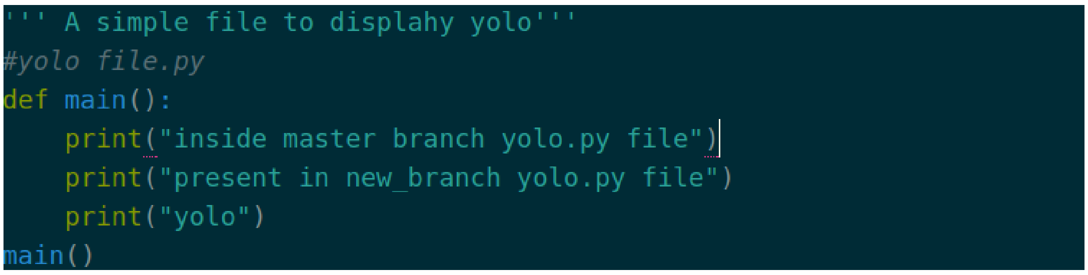
```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

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

        both modified:   yolo.py
```

Now to fix merge conflicts we open, the file that we modified,



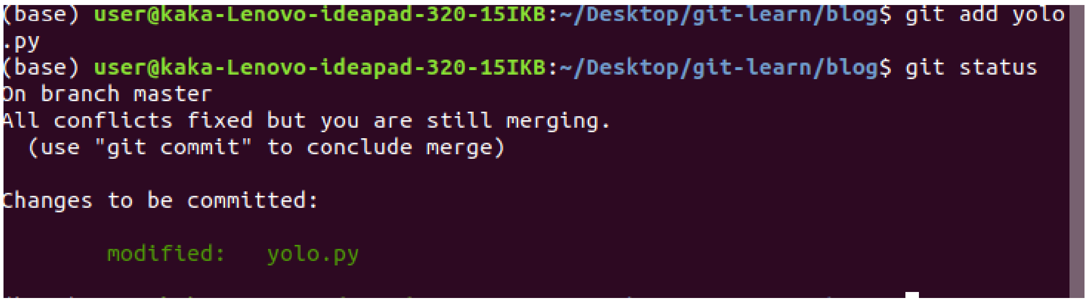
```
''' A simple file to displahy yolo'''
#yolo file.py
def main():
<<<<<<< HEAD
    print("inside master branch yolo.py file")
=======
    print("present in new_branch yolo.py file")
>>>>>>> new_branch
    print("yolo")
main()
```

Now we see the modification we need to do,



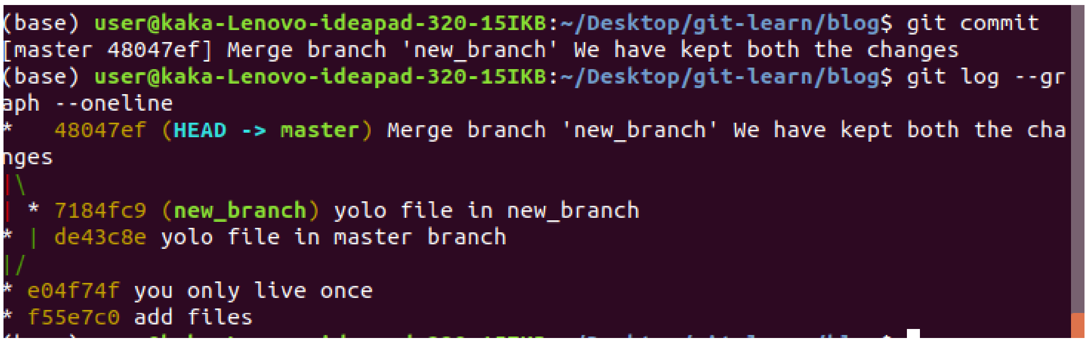
```
''' A simple file to displahy yolo'''
#yolo file.py
def main():
    print("inside master branch yolo.py file")
    print("present in new_branch yolo.py file")
    print("yolo")
main()
```

now we add the file and commit it to see that the conflict has been resolved,



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git add yolo
.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

        modified:   yolo.py
```

then we call git commit to complete the resolve, along with git log to see the changes done.



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git commit
[master 48047ef] Merge branch 'new_branch' We have kept both the changes
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog$ git log --gr
aph --oneline
*   48047ef (HEAD -> master) Merge branch 'new_branch' We have kept both the cha
nges
|\
| * 7184fc9 (new_branch) yolo file in new_branch
* | de43c8e yolo file in master branch
|/
* e04f74f you only live once
* f55e7c0 add files
```

git log –graph –oneline = [This shows a summarized view of the commit history for a repo (https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History).](https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History)

7. What is GitHub ?

GitHub is a web-based Git repository hosting service. On top of the version control functionality of Git, GitHub includes extra features like bug tracking, wikis, and task management. GitHub lets us share and access repositories on the web and copy or clone them to our local computer, so we can work on them. GitHub is a popular choice with a robust feature set, but it's not the only one. Other services that provide similar functionality are Bit-bucket, and Git-Lab.

Follow the workflow at **https://github.com/join (https://github.com/join)** to set up a free account, username, and password. After that, **these steps (https://help.github.com/articles/create-a-repo/)** will help you create a brand new repository on GitHub.

1.Workflow on GitHub,

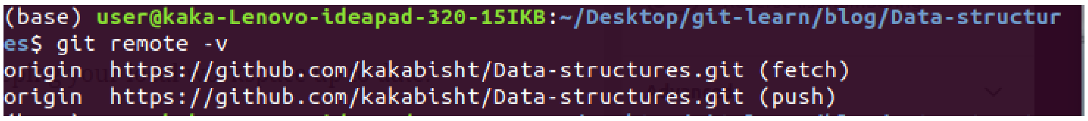
1. git clone https:gitrepo.git , to copy the GitHub files into local repository
2. you now have a local repository, where you modify code
3. then you add and commit the files
4. git pull is used to fetch the updates happened on GitHub remote repository.
5. then you push the file using git push

This can be useful for keeping your local workspace up to date.

- **https://help.github.com/en/articles/caching-your-github-password-in-git (https://help.github.com/en/articles/caching-your-github-password-in-git)**

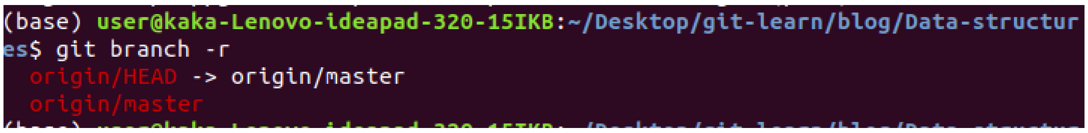
- **https://help.github.com/en/articles/generating-an-ssh-key (https://help.github.com/en/articles/generating-an-ssh-key)**

When we call a git clone to get a local copy of a remote repository, Git sets up that remote repository with the default origin name



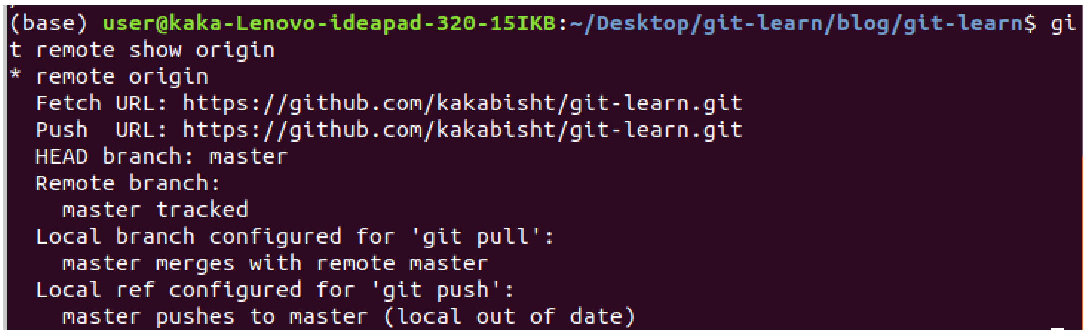
```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/Data-structur
es$ git remote -v
origin  https://github.com/kakabisht/Data-structures.git (fetch)
origin  https://github.com/kakabisht/Data-structures.git (push)
```

Whenever we're operating with remotes, Git uses remote branches to keep copies of the data that's stored in the remote repository. We could have a look at the remote branches that our Git repo is currently tracking by running git branch -r



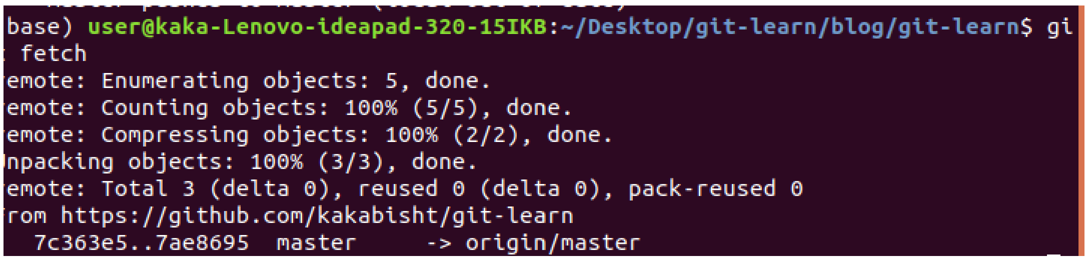
```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/Data-structur
es$ git branch -r
  origin/HEAD -> origin/master
  origin/master
```

git remote show origin will tell if our git repository has been updated, and now we have to pull data from git repository.



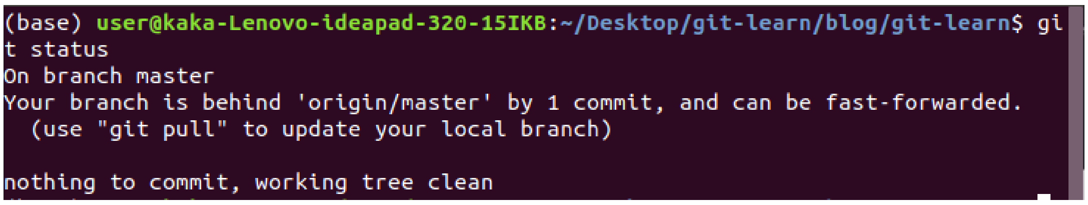
```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ gi
t remote show origin
* remote origin
  Fetch URL: https://github.com/kakabisht/git-learn.git
  Push  URL: https://github.com/kakabisht/git-learn.git
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (local out of date)
```

we use git fetch



```
base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ gi
 fetch
emote: Enumerating objects: 5, done.
emote: Counting objects: 100% (5/5), done.
emote: Compressing objects: 100% (2/2), done.
npacking objects: 100% (3/3), done.
emote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
rom https://github.com/kakabisht/git-learn
  7c363e5..7ae8695  master     -> origin/master
```

if then we use git status, it tells our local repo is behind the git repo.



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ gi
t status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean
```

git fetch fetches remote updates but doesn't merge; git pull fetches remote updates and merges.

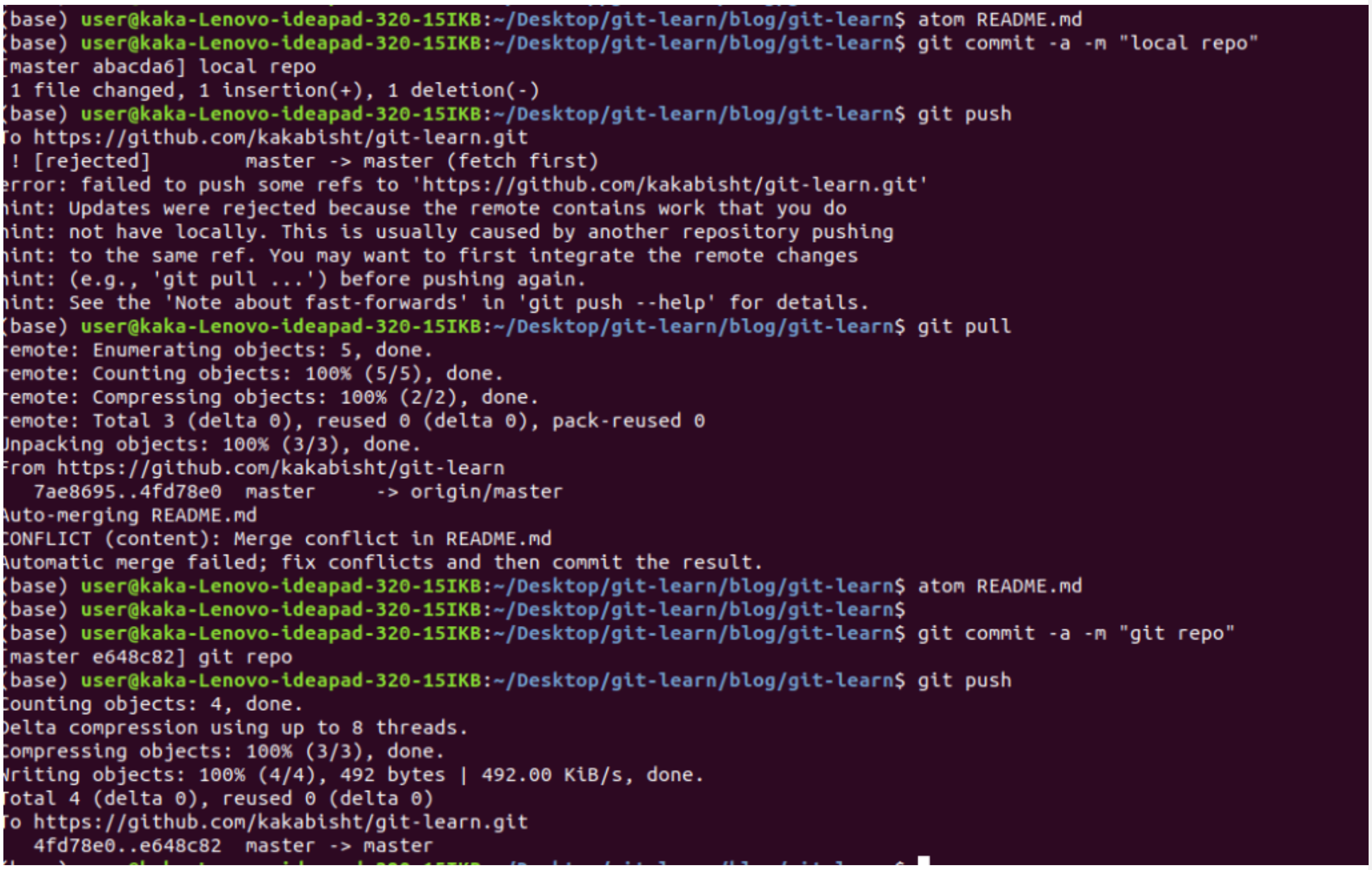
1. git remote update : **Fetches the most up-to-date objects (https://git-scm.com/docs/git-remote#Documentation/git-remote.txt-emupdateem)**
2. git branch -r : **Lists remote branches (https://git-scm.com/docs/git-branch#Documentation/git-branch.txt--r)**; can be combined with other branch arguments to manage remote branches

3. >>> -> conflict markers

2.The Pull-Merge-Push Workflow :

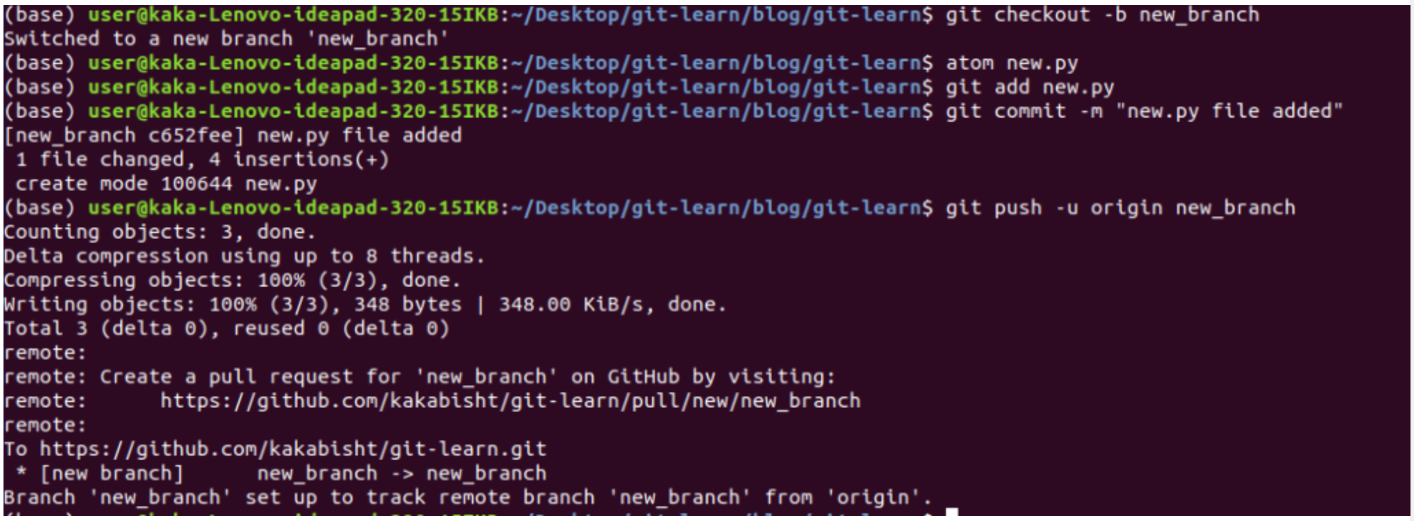
Here we use the three way merge, for example if we modified code and were pushing the code but our git remote repository is also modified it will show an error in pushing. To solve that error,

1. We need to use git pull
2. check for any merge conflicts
3. then add and commit the files
4. then finally push the file.



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ atom README.md
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git commit -a -m "local repo"
[master abacda6] local repo
 1 file changed, 1 insertion(+), 1 deletion(-)
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git push
o https://github.com/kakabisht/git-learn.git
 ! [rejected]        master -> master (fetch first)
rror: failed to push some refs to 'https://github.com/kakabisht/git-learn.git'
int: Updates were rejected because the remote contains work that you do
int: not have locally. This is usually caused by another repository pushing
int: to the same ref. You may want to first integrate the remote changes
int: (e.g., 'git pull ...') before pushing again.
int: See the 'Note about fast-forwards' in 'git push --help' for details.
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git pull
emote: Enumerating objects: 5, done.
emote: Counting objects: 100% (5/5), done.
emote: Compressing objects: 100% (2/2), done.
emote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
npacking objects: 100% (3/3), done.
rom https://github.com/kakabisht/git-learn
   7ae8695..4fd78e0  master     -> origin/master
uto-merging README.md
ONFLICT (content): Merge conflict in README.md
utomatic merge failed; fix conflicts and then commit the result.
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ atom README.md
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git commit -a -m "git repo"
[master e648c82] git repo
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git push
ounting objects: 4, done.
elta compression using up to 8 threads.
ompressing objects: 100% (3/3), done.
riting objects: 100% (4/4), 492 bytes | 492.00 KiB/s, done.
otal 4 (delta 0), reused 0 (delta 0)
o https://github.com/kakabisht/git-learn.git
   4fd78e0..e648c82  master -> master
```

when we push a branch to the repo we need to add, a few parameters on the push command, git push -u origin branch_name.



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git checkout -b new_branch
Switched to a new branch 'new_branch'
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ atom new.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git add new.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git commit -m "new.py file added"
[new_branch c652fee] new.py file added
 1 file changed, 4 insertions(+)
 create mode 100644 new.py
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git push -u origin new_branch
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 348 bytes | 348.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'new_branch' on GitHub by visiting:
remote:      https://github.com/kakabisht/git-learn/pull/new/new_branch
remote:
To https://github.com/kakabisht/git-learn.git
 * [new branch]      new_branch -> new_branch
Branch 'new_branch' set up to track remote branch 'new_branch' from 'origin'.
```

3.Rebasing

Rebasing means changing the base commit that's used for our branch

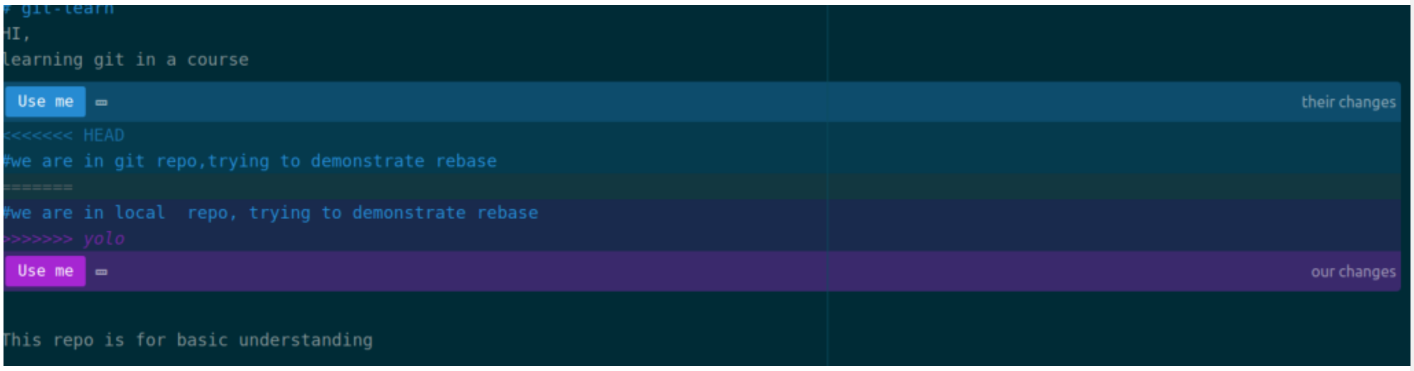
The problem with three way merges is that because of the split history, it's hard for us to debug when an issue is found in our code, and we need to understand where the problem was introduced. By changing the base where our commits split from the branch history, we can replay the new commits on top of the new base. This allows Git to do a fast forward merge and keep history linear.

We run the command git rebase, followed by the branch that we want to set as the new base. When we do this, Git will try to replay our commits after the latest commit in that branch. This will work automatically if the changes are made in different parts of the files, but will require manual intervention if the changes were made in other files.

example : If we were working on the main branch and so was our co-worker, he might have added changes different to ours. Hence we,

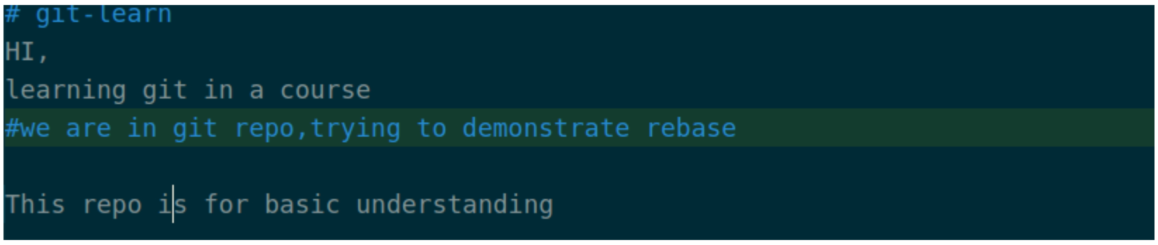
1.git fetch the recent changes done in GitHub repository

2.we try to rebase, in which it shows merge conflicts



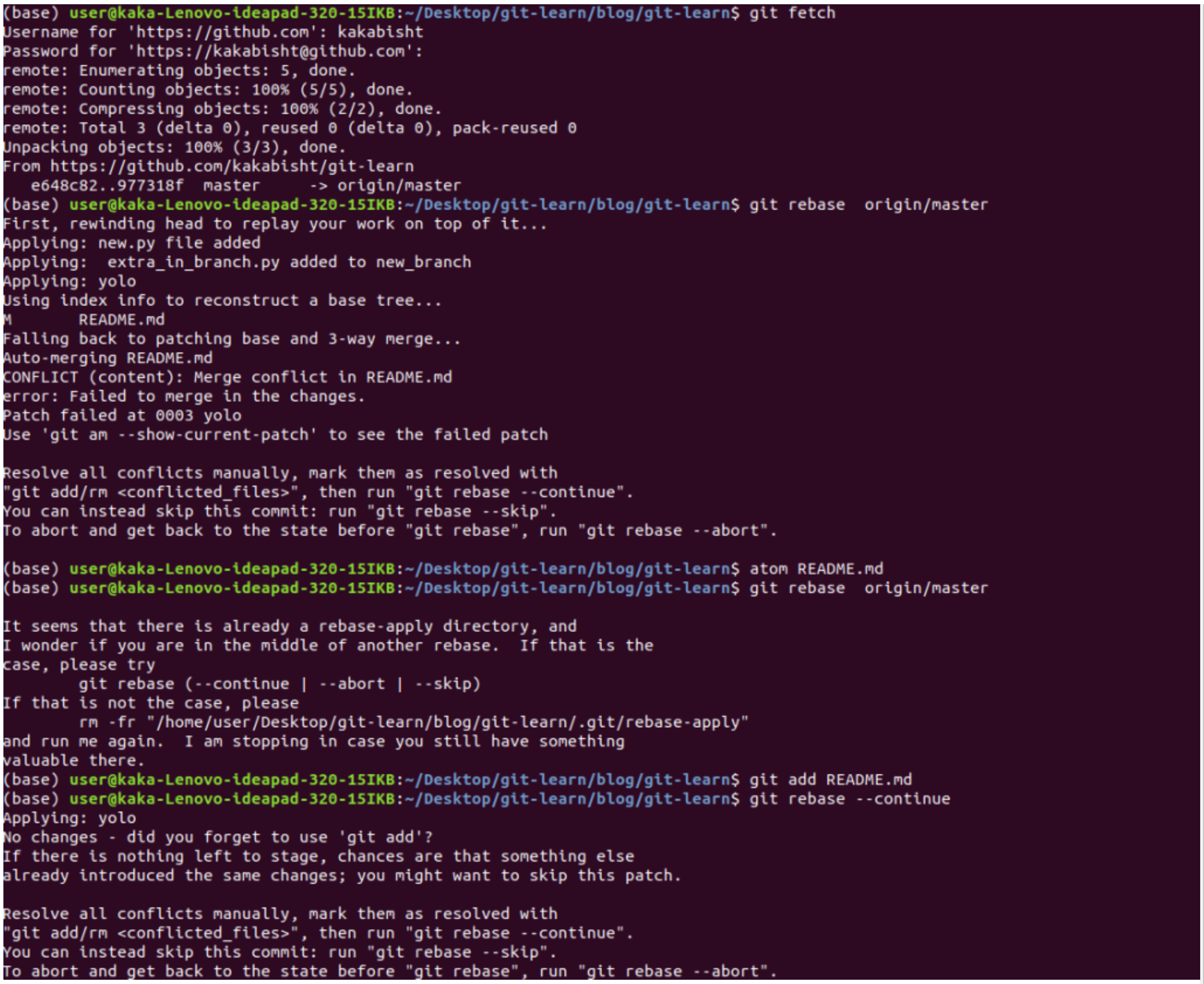
```
# git-learn
HI,
learning git in a course
<<<<<<< HEAD
#we are in git repo,trying to demonstrate rebase
=======
#we are in local  repo, trying to demonstrate rebase
>>>>>>> yolo

This repo is for basic understanding
```

3.we resolve the merge conflicts, then add the file



```
# git-learn
HI,
learning git in a course
#we are in git repo,trying to demonstrate rebase

This repo is for basic understanding
```

4.we add the file again, and use git rebase –continue.



```
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git fetch
Username for 'https://github.com': kakabisht
Password for 'https://kakabisht@github.com':
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/kakabisht/git-learn
   e648c82..977318f  master     -> origin/master
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git rebase  origin/master
First, rewinding head to replay your work on top of it...
Applying: new.py file added
Applying:  extra_in_branch.py added to new_branch
Applying: yolo
Using index info to reconstruct a base tree...
M       README.md
Falling back to patching base and 3-way merge...
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
error: Failed to merge in the changes.
Patch failed at 0003 yolo
Use 'git am --show-current-patch' to see the failed patch

Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".

(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ atom README.md
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git rebase  origin/master

It seems that there is already a rebase-apply directory, and
I wonder if you are in the middle of another rebase.  If that is the
case, please try
        git rebase (--continue | --abort | --skip)
If that is not the case, please
        rm -fr "/home/user/Desktop/git-learn/blog/git-learn/.git/rebase-apply"
and run me again.  I am stopping in case you still have something
valuable there.
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git add README.md
(base) user@kaka-Lenovo-ideapad-320-15IKB:~/Desktop/git-learn/blog/git-learn$ git rebase --continue
Applying: yolo
No changes - did you forget to use 'git add'?
If there is nothing left to stage, chances are that something else
already introduced the same changes; you might want to skip this patch.

Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".
```

Merge conflicts are not uncommon when working in a team of developers, or on Open Source Software. Fortunately, GitHub has some good documentation on how to handle them when they happen:

1. **https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-merge-conflicts (https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-merge-conflicts)**
2. **https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line (https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line)**

4.Collaboration

Forking is a way of creating a copy of the given repository so that it belongs to our user. In other words, our user will be able to push changes to the forked copy, even when we can't push changes to the other repo. When collaborating on projects hosted on GitHub,

A pull request is a commit or series of commits that you send to the owner of the repository so that they incorporate it into their tree.

1. A typical pull request workflow,

1. fork the repo, you want to work on.
2. clone in on the local repo, by git clone.
3. create a new branch, work with modifications in a new branch
4. add and commit the changes made,
5. git push -u origin files
6. the create a pull request, by going on to the GitHub web interface and click on compare & pull request .

Now if we have to modify our pull request ,then

1. we open the file, apply the changes
2. we add and commit the file
3. we push the file
4. then in our GitHub interface, we can see our new as well old pull request there.

Now what to do if you are supposed to squash you changes in a single commit,

1. we will call git rebase -i master : it shows a list of all the commits
2. The default action here is pick which takes the commits and rebases them against the branch we selected
3. we have two options, I recommend squash
4. squash : two commits are added together, open up a text editor to edit
5. fix up : discards the old commit
6. after this we are given a text file to edit, as our final squashed commit
7. save and exit from the commit editor
8. Now we don't want to merge our branches, but modify our previous commit history hence we push forcefully

2.Code reviews

Doing a code review means going through someone else's code, documentation or configuration and checking that it all makes sense and follows the expected patterns. The goal of a code review is to improve the project by making sure that changes are high quality. It also helps us make sure that the contents are easy to understand. That the style is consistent with the overall project. And that we don't forget any important cases.

Nit – a minor bug

Style guide : pep8 for python, for more info go through these.

1. **http://google.github.io/styleguide/ (http://google.github.io/styleguide/)**
2. **https://medium.com/osedea/the-perfect-code-review-process-845e6ba5c31 (https://medium.com/osedea/the-perfect-code-review-process-845e6ba5c31)**
3. **https://smartbear.com/learn/code-review/what-is-code-review/ (https://smartbear.com/learn/code-review/what-is-code-review/)**

Now some resources from which I gained my knowledge,

1. **https://git-scm.com/doc (https://git-scm.com/doc)**
2. **https://www.coursera.org/learn/introduction-git-github/home/welcome (https://www.coursera.org/learn/introduction-git-github/home/welcome)**

GIT WITHOUT PUNS

~~Git~~ Got questions?

29

reach me out at twitter

Tags: Branches, GIT, GitHub, Merge, Ubuntu, Version Control System **Categories: Uncategorized**

Published by Hridyesh singh bisht

Hi , i am Hridyesh Singh Bisht . Currently a sophomore at Symbiosis University , An Athlete, A Geek, and A musician. **View all posts by Hridyesh singh bisht**