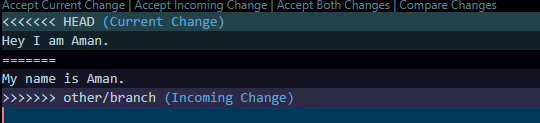
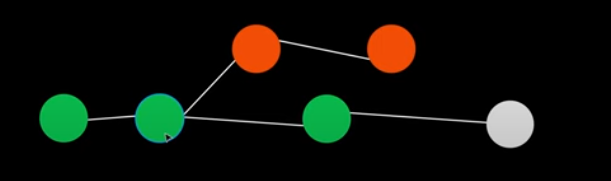
**Git:** Git is used to manage individual’s code, we can manage and share our code and we can also mark checkpoints in our code. It is used to track changes in code over time. Git operates locally means all the changes will be saved in our computer.

**GitHub:** GitHub is used when we have many people and we want to merge everybody’s code into a single file without sharing the code with other tools.

1. Downloading:
   1. Download [Git](https://git-scm.com/downloads) from here.
2. Setup Git locally:
   1. git config –global user.name “Aman” - **This is the username everybody will see**
   2. git config –global user.email “[aman@gmail.com](mailto:aman@gmail.com)” – **This is the email address displayed to everyone**
   3. git config –global core.autocrlf “input” – **This is used when code will not work**
   4. git config –global core.editor “code –wait” – **If code is not correct then VS Code or any other code editor will open**
3. Making Git available:
   1. To make git available in our project then click on **Source Control** from the left side on third number and then click on **initialize repository**.
4. Stages:
   1. U – Untracked 🡪 Not tracked by GitHub.
   2. A – Added or staged 🡪 Managed by GitHub.
   3. C – Committed 🡪 Added to GitHub repository.
   4. M – Modified 🡪 Added new content to the file.
5. Checkpoint:
   1. Create a file of any type you want.
   2. Click on source control tab and click on staged changed and click on “+” to add into staged changes.
   3. Then type any type of message related to work you did till now and click on commit.
   4. This will add a checkpoint to your code and now you can do your further work.
   5. Then again you can do the same thing to mark another checkpoint into you file.
   6. **To see the checkpoints of your file: use this command 🡪 git log –oneline**
      1. The checkpoints will have unique ID, this can be used to manage the checkpoints.
6. Git Ignore:
   1. Git ignore is used when we want to keep that file but don’t want GitHub to keep the track of the file.
   2. To do this we create a file by name 🡪 **.gitignore**
   3. Then we write the name of those file which we want to ignore into the gitignore file.
7. Going back to the saved checkpoints:
   1. To go back and delete all the data of previously done work.
   2. Use this command: **git reset --hard HEAD~1**
   3. **1 means how many stages we want to go back**.
   4. For example: we have created a gitignore file and added test.txt into it and committed it.
   5. After running this command - git reset --hard HEAD~1 🡪 our created file gitignore will be deleted and we are now 1 stage/step back.
8. Some more commands:
   1. To check status of our code folder 🡪 **git status –v** 🡪 This will list all the stages of all the files, like – U, A, M, etc.
9. Branching:
   1. Branching means not adding new code directly into the main file/code, instead of this we create a new branch and provide the main and then changes are made into that branch and then finally we merge the both and our new main branch is ready.
   2. In branching if we add a new line of code in main and another line of code in other branch and but some line of code is different in both of them then we will get a **conflict error** means git is now confused which he has to take.
   3. After conflict it will give 3 options while merging the code:
      1. Accept the code form the other branch.
      2. Accept the code from the main branch.
      3. Accept both code.
   4. To create a new branch: **git branch “branch name”.**
   5. To check the number of branch: **git branch.**
   6. To switch b/w branches: **git “branch name”.**
   7. To merge branches: **git merge “branch name”** (you need to be on main branch)
   8. Merging Conflict:
      1. We have three option and we can choose anyone.

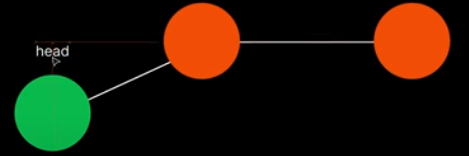


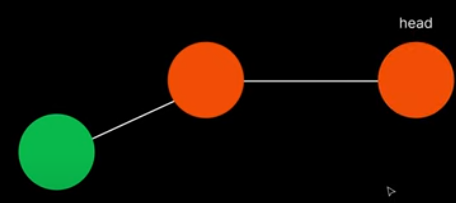
* 1. Merging Techniques:
     1. Three way merge



Original code is in 2nd green circle and it has one copy in 1st orange circle and some code added in both copied circles and the final code is in the white circle (merged from 3rd green circle, 2nd orange circle and 2nd green circle) this is called **Three way merging technique**.

* + 1. Fast Forward merging:





In this the **location of our head branch is changed**. It looks like that we have merged the code but actually is not merged only **the location of our head branch is changed**.

* 1. Deleting branch:
     1. For deleting branch: **git branch –d “branch name”**.

1. Stashing:
   1. Whenever we write some lines of code and we try to move in another branch without committing the code then there will be error message and it will ask to either commit the code or if you move to another branch then it will delete the newly added code or you can keep the code as draft.
   2. To move into another branch without deleting the newly added code we have to use the stashing. **This command will remove the new changes but keep those as draft in its memory.**
   3. Stashing command: **git stash**.
   4. After this we can move into another branch can do our work there.
   5. Now coming to our main branch we can get our drafted code.
   6. To get our code back: **git stash apply**.