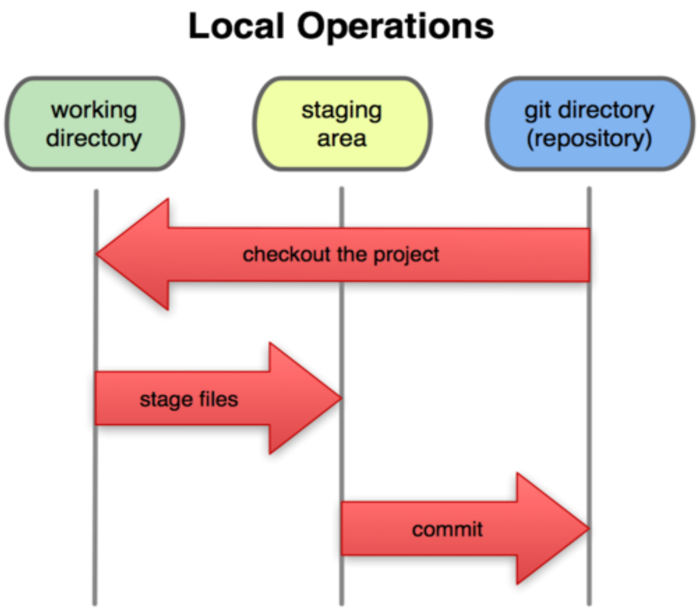
GIT/GITHUB

- Git is a ***version control system***, which helps to:
    1. Roll back to previously working state.
    2. Files can be pulled and pushed with a repository for group of people to work
    3. Captures snapshots
    4. Almost every operation is local and then you can push your repositories to the centralised server
    5. Git has integrity. Git uses checksum so that files are not tampered with while downloading. Every file has a unique SHA checksum (a string like 544bf79h) and it should match with the downloaded file to know that there's no virus or the data is not hacked.
    6. Git generally only adds data i.e. your repositories just grow

- Github is the repository hosting service website
- Created by Linus Torvalds in 2005, free of charge
- cd command is for changing the directory of the command
- GUI is a easier user interface for pulling and pushing files.
- When you commit and make changes your name and email is visible so that a group can see where the error was made or who modified the programs and you can be contacted

GIT THREE STAGE ARCHITECTURE:



- Suppose if you're working on a file and you introduce an error and you want to come back on the version 1, git makes this possible by taking snapshots of the time when you were working on version 1 (almost like capturing a moment like a photograph), this is called to ***commit***.

- Working directory is where you're directly working on your computer, the file, folder, website etc.
- Staging Area are the files which you want to enter the next commit. eg. suppose we have three files like index.html, engine.js and index.css. You came across an error in engine.js so you sent the other two to the staging area (pushed to the repository) and the left one can be commited later.
- Git repository is a hidden directory (.git) and keeps your file compressed here

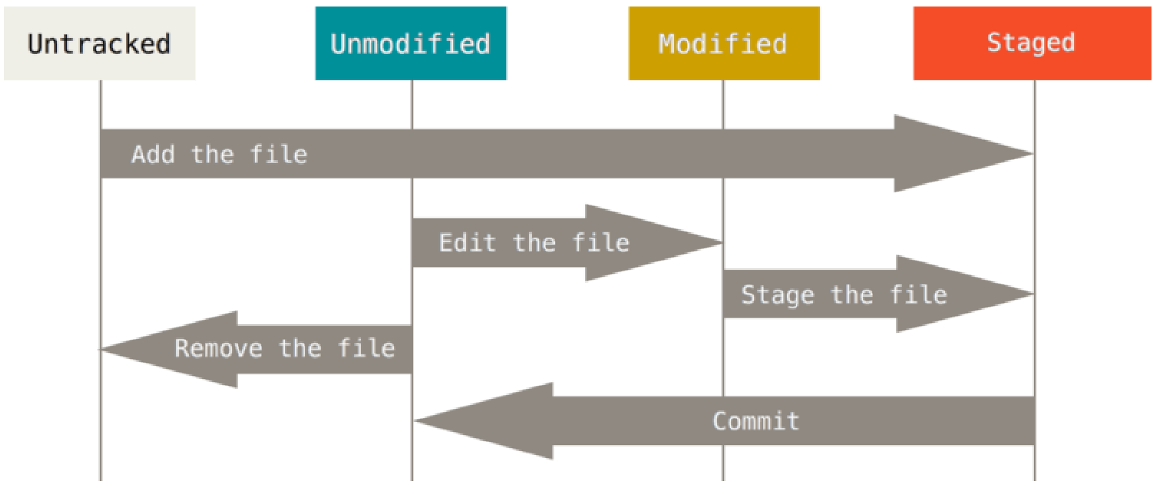
COMMANDS:

1. *git status*- to see if you're in the repository or not
2. *git init*- to make or initialise a new directory (.git)
3. *git add –a* OR *git *dot** -adds all the files to the repository (staging area, a snapshot is taken)
4. *git commit -m* – commits the files and it shows that the working tree is clean
5. *git log* – shows all the commits done by you along with your name and everything like a snapshot
6. *git add *file name** - to commit a single file
7. *git restore *file name** - if you don't want to save the changes done in the file
8. *rm -rf .git*- this deletes the .git repository in which you have your current files (a linux based command)
9. *pwd-* to come in the present work directory (linux command)
10. *ls* – lists all the files in the pwd
11. *cd *filename** - to go into some directory
12. *git clone *url/address of the repository to be cloned*-* to clone a directory
13. *q* – to quit the log
14. *touch *file name*-* to make a new file through the terminal
15. *touch .gitignore* – makes a file in which you can put files which are to be ignored. Now even if you make changes in this file, it will not be reflected in the repository
16. **.log(or any other file extension you want)-* suppose there are many copies or types of log file which you want to ignore therefore instead of typing them manually, we can use this command to ignore any file which has a .log extension. You can use it for any file. We can use it for folders too. For outside folder / after name, for the inside /filename/ . We can also add path to ignore.
17. *git diff*- compares working area to staging directory. (working directory edit shown in red and staging area in green) now if you use get add and then use this it will not show anything showing that everything in the staging area ready to commit.
18. *git diff –staged* – the entire process between two commits that is compares the last commit to your staging area.
19. git commit -a -m – Stages all the tracked files(unmodified, staged or modified files) and sends them to commit but if you have a untracked file it doesn't go to the commit, ignoring the staging area.

20. *git mv *original file name* *renamed file name**- To rename a pre existing file and add it directly to the staging area.
21. *git rm –cached *filename** - this will untrack the file which was being tracked at some point, for eg. files which were tracked and then added to the gitignore afterwards, as they have been snapshotted before, they need to be untracked in order for them to stay ignored if any kind of changes are made in them and hence does not reflect in the status.
22. *git log -p* -shows what exactly is commited recently and by whom. We can also use git log -p -3 or any number and it shows that number of commits
23. *git log –stat* – shhows a small summary/statistics of all the edits and commits
24. *git log –pretty=oneline* – shows all the commits summarised respectively in a single line. Makes it easy for printing on a paper.
25. *git log –pretty=short* – short summary of the commit, can also be pretty=long
26. *git log --since2.days* – commits of last 2 days. can also be weeks, months etc.
27. *git log –pretty=format:" %h -- %an"* – this shows the commit hash with the author's name. we can use many placeholders in format with different functions. The string comma is mandatory
28. *git commit* –amend – to amend any comments of the commits made by other people mostly and then merge the original with yours. A wim editor window opens up usually so to add your comment press 'i' button and then write your comment. Then press ESC then :WQ and it closes the window after your amendment.
29. *git restore –staged *filename** - to unstage a file. Almost like removing after adding to commit.
30. *git checkout -- *filename** - to unmodify a file. Helps when you've accidently deleted your data. We use git checkout -f if there a bunch of files that need unmodification
31. *git remote add origin *url**- for connecting github to git. *name of url* is usually witten as origin.
32. *git remote*- shows what remote repository you have
33. *git remote -v* – to check the pull, push directory
34. *git push -u origin master* – for pushing file to repository
35. *git remote set-url origin *url** - to switch repositories
36. *git push -d origin *branchname**   - this deletes the branch from the repository
37. *git config --global alias.st status* - for giving alias to any big command into a short one. In this case we can obtain status by typing st.
38. *git config --global alias.unstage 'restore --staged'* – alias to config -staged
39. *git config --global alias.last 'log -p -1'* –   alias for log -p
40. *git checkout -b develop* - this is used to create a side develop branch. We can name it anything other than develop. After this command whatever changes are made in the files will not affect the master branch.
41. *git checkout master* - to go back to the master branch
42. *git checkout *branchfile name**-   to go back to the branch
43. *git branch*- list of all the branches that are made
44. *git branch -v* - we get last commit info including hash and message
45. *git branch –merged* – shows all the branches which have been merged

46. *git branch –no-merged* - which have not been merged
47. *git branch -d *filename** - deletes the branch but will give error in deleted if have not been merged. A confirmation therefore a new command is needed showing that you're not accidently deleting the branch. So in this case -D is used.

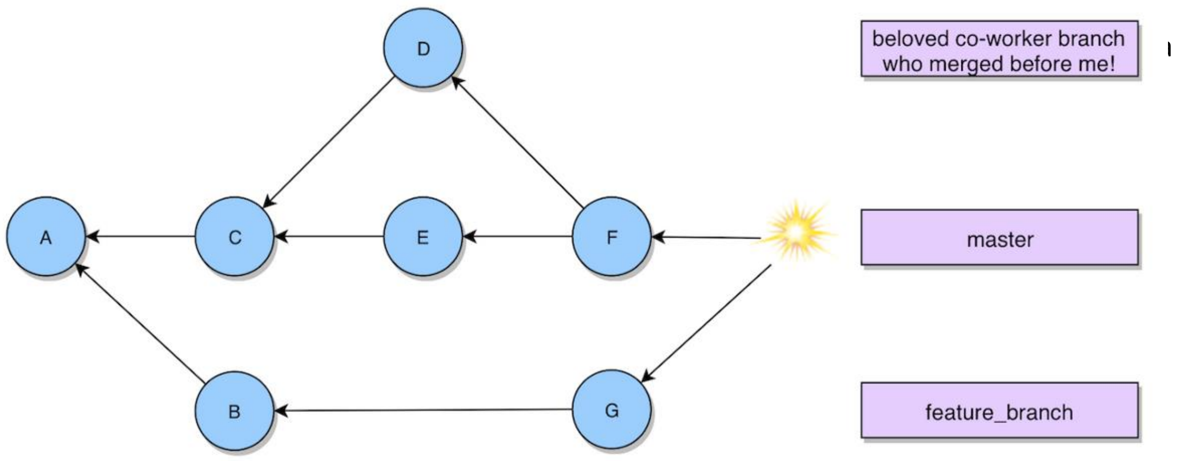
FILE STATUS LIFECYCLE:



- when *get add –a* is used then all the *Untracked* file comes to the staging area i.e they become *Unmodified*.
- Now if any of the files is edited, it becomes *Modified*
- Now again if they're added through git add it becomes *Staged*.

CREATING & SWITCHING BRANCHES:

- The main branch in the repository is called *Master* branch.
- A *side branch* k/a *Develop Branch* is something which can be worked on without affecting the master branch.
- The branches can be merged depending upon the choice. eg: group of friends making a website and everyone doing their own thing and merging when they all like it.



- <<<<< these are known as conflict resolution markers
- A conflict can be because of a issue that cannot be resolved by the computer like choosing a main heading which may have been edited many times in the side branch. As the heading is an important aspect, and therefore git makes the user to choose and resolve the conflict.
- git add *filename* saves the branch which filename you chose

BRANCHING WORKFLOW:

- There are 2 kinds of branching workflow:
    1. Long Running Branches- Which stay till the end of a particular project
    2. Topic Branches- Short lived branch which only exists till a particular topic is worked upon.