**GIT (global information tracker)**

**Version Control:**

This is the process of maintaining multiple versions of the code. All the team members upload their code (check in) into the remote version controlling system. The Version control system accepts the code uploads from multiple team members and integrates it, so that when the other team members download the code, they will be able to see the entire work done by the team.

Version control system also preserve older and later versions of the code so that at any time, we can switch between whichever version we want. Version control system also keep a track of who is making what kind of changes

**Version control systems are categorized into 2 types**

1. Centralized version controlling

2. Distributed version controlling

**Centralized Version controlling:**

Here we have a remote server (code repository) into which all the team members check in the code and all the features of version controlling are implemented in this remote server. E.g: SVN (Subversion)

**Distributed version controlling**:

Here we have a local repository installed on every team members machines where version controlling happens at the level of individual team members form where it is uploaded into a remote server where version controlling happens for the entire team.

E.g: GIT

GIT:

Git is a distributed version controller system

Git is a Tree History storage system

Git is a content tracking management system

**Setting up git on Windows:**

1. Download git from

https://git-scm.com/downloads

2. Install it

3. Open git bash and execute the git commands

**Setting up git in ubuntu Linux servers:**

1 Update the apt repository

sudo apt-get update

2 Install git

sudo apt-get install -y git

**Configuring user and email globally for all users on a system:**

git config --global user.name "sscademy"

git config --global user.email “[sscademy@gmail.com](mailto:sscademy@gmail.com)”

**On the local machine git uses three sections**

1 Working directory

2 Staging Area

3 Local repository

Working directory is the location where all the code is created. Initially all the files present here are called as untracked files.

Staging area is the location where file indexing happens and it is the buffer area of git and the files are called as indexed files

Local repository is where version controlling happens and the files are called as committed files

**Start working with GIT:**

1. To initilise the working dir as a git repo

**$git init**

This will create a hidden folder called .git where it will store all configurations for git to work

2. To send a file into from working dir to stagging area

**$git add filename**

To send multiple files

**$git add file1 file2 file3**

To send all files

**$git add .**

. represents current working dir

3 To bring files from stagging area to working dir

**$git rm --cached filename**

(or)

**$git reset filename**

4 To send files from stagging area to local repository

**$git commit -m "Some msg"**

5 To check the status of working dir and stagging area files

**$git status**

6 To see the commits done on the local repository

**$git log**

To see this output in simple one line format

**$git log –oneline**

**.gitignore:**

This is a special configuration file that is used to store private files info.Any file whose name is stored in .gitignore will not be accessed by git

1. Create few file

**$touch file1 file2 file3 file4**

2. Check the git status

**$git status**

It will show the above 4 files as untracked

3. Create .gitignore and store the above 4 filenames in it

cat > .gitignore

file1

file2

file3

file4

4. Check the status of git

It will no longer show file 1-4

**Branching in Git:**

This is a feature of git using which we can create separate branches for different functionalities and later merge them with the main branch also known as the master branch. This will help in creating the code in an uncluttered way

1. To see the list of local branches

**$git branch**

2. To see the list all branches local and remote

**$git branch -a**

3. To create a branch

**$git branch branch\_name**

4 To move into a branch

**$git checkout branch\_name**

5. To create a branch and also move into it

**$git checkout -b branch\_name**

6. To merge a branch

**$git merge branch\_name**

7. To delete a branch that is merged

**$git branch -d branch\_name**

This is also called as soft delete

8. To delete a branch that is not merged

**$git branch -D branch\_name**

This is also known as hard delete

**Note:** Whenever a branch is created whatever is the commit history of the parent branch will be copied into the new branch

**Note:** Irrespective of, on which branch a file is created or modified git only considers form which branch it is committed and the file belongs to that committed branch only.

Github: At a high level, GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code.

Working with GIT:

1. GIT CLONE: This will download all the code from the remote repository into the local repository and it is generally used only once when all the team members want a copy of the same code

Syntax: git clone remote\_git\_repo\_url

1. GIT FETCH: This will download only the modified files but it will place them on a separate branch called as "remote branch", we can go into this remote branch check if the modifications are acceptable and then merge it with the main branch.
2. GIT PULL: This will download only the modified files and merge them with our local branches
3. GIT MERGE: Merging always happens bases on the time stamps of the commits

e.g:

a. Create few commits on master

touch f1

git add .

git commit -m "a"

touch f2

git add .

git commit -m "b"

b Check the git commit history

git log --oneline

c Create a test branch and create few commits on it

git checkout -b test

touch f3

git add .

git commit -m "c"

touch f4

git add .

git commit -m "d"

d. Check the commit history

git log --oneline

e. Go back to master and create few more commits

git checkout master

touch f5

git add .

git commit -m "e"

touch f6

git add .

git commit -m "f"

f. Check the commit history

git log --oneline

g. Merge test with master

git merge test

h. Check the commit history

git log –oneline

1. GIT REBASE: This is called as fast forward merge where the commits coming from a branch are projected as the top most commits on master branch.

* Implement step1-6 from above scenario
* To rebase test with master

git checkout test

git rebase master

git checkout master

git merge test

* Check the commit history

git log –oneline

1. GIT CHERRYPICK: This is used to selectively pick up certain commits and add them to the master branch

E.g:

1 On master create few commits

a--->b

2 Create a test branch and create few commits

git checkout -b test

a--->b--->c--->d--->e--->f--->g

3 To bring only c and e commits to master

git checkout master

git cherry-pick c\_commitid e\_commitid

1. GIT RESET: This is a command of git using which we can toggle between multiple versions of git and access whichever version we want

* Reset can be done in 3 ways
  + 1. Hard reset
    2. Soft reset
    3. Mixed reset
* In hard reset HEAD simply points to an older commit and we can see the

data as present at the time of that older commit

* + - 1. Create few commits on master

a-->b--->c

* + - 1. To jump to b commit from c

git reset --hard b\_commit\_id

* Soft reset will also move the head to an older commit but we will see the condition of the git repository as just one step prior to the c commit ie the files will be seen in the stagging area

Syntax: git reset --soft b\_commitid

* Mixed reset also moves the head to an older commit but we will see the condition of git as 2 steps prior to the c commit i.e. the files will be present in the untracked/modified section

Syntax: git reset --mixed b\_commitid

1. GIT TAGGING: This is used to place book marks on important commits. Generally it used to identfy the commits that are related to release events

* Tags are classifed into 2 types
  + - * 1. Light weight tags
        2. Annoted Tags
    1. Light weight tags: These tags only have a tage name To create a light weight tag to the latest commit

Syntax: git tag some\_tag\_name

* + - To create a light weight tag to an older commit

Syntax: git tag some\_tag\_name older\_commit\_id

* + - * To create an annoted tag to the latest commit

Syntax: git tag -a "some\_tag\_name" -m "some msg related to the tag"

* + - To create an annoted tag for an older commit

Syntax: git tag -a "some\_tag\_name" -m "some msg related to the tag" older\_commit\_id

* + - To see the list of all tags

Syntax: git tag

* + - To push the tags into remote github

Syntax: git push --tags

* + - To delete a tag locally

Syntax: git tag -d tagname

1. GIT STASHING: Stash is a section of git into which once the files are pushed git cannot access them.
   * + To stash all the files present in the staging area

Syntax: git stash

* + - To stash all files, present in stagging area and untracked section

Syntax: git stash -u

* + - To stash all files present in stagging area, untracked section and .gitignore

Syntax: git stash -a

* + - To see the list of stashes

Syntax: git stash list

* + - To unstash a latest stash

Syntax: git stash pop

1. GIT SQUASH: This is the process of merging multiple commits and making it look like a single commit.This can be done using the git rebase command
   * + Create a commit history

a --> b --> c --> d --> e --> f

HEAD is pointing to f commit

Note: a commit is called as the "initial commit" and it cannot be squashed

In the above scenario we can squash only a max of 5 commits

* + - * To squash

git rebase -i HEAD~5

This will open the top 5 commits in vi editor For which ever commits we want to perform a squash opration remove the word "pick" and replace it with "squash"

* + - * Check the commit history

git log –online