What is Git?

* Git is a central repository using which we can manage our project source code
* Git is also called it a version controlling system
* It maintains all modifications happening to a specific file
* Because of versions troubleshooting and fixing bugs is easy
* If something goes wrong in current version, we can rollback to previous version
* Records who modified, when it is modified(timestamp) and why it is modified
* Git is distributed version controlling system
* Git is fast when it is compared with other version controlling tools
* Multiple developers can easily collaborate and work on same project
* It also works as backing up our project code

Git terminology:

Git is a client server architecture.

Client ===>> Git bash and server ===>> GitHub

Repository: group of project files to store one single area and each project has one repository. ==>>> GitHub has ==>> n. Number of repositories.

Cloning: remote repository to getting the local ==>>> cloning ==>>> git clone URL (each project has one URL) ===>> locally

Fork: projects are copied from one account to another account.

Local repository: getting the remote repository to our local laptop. Git clone clones the remote copy into our local machine.

**Note:** The local copy is called as local repository

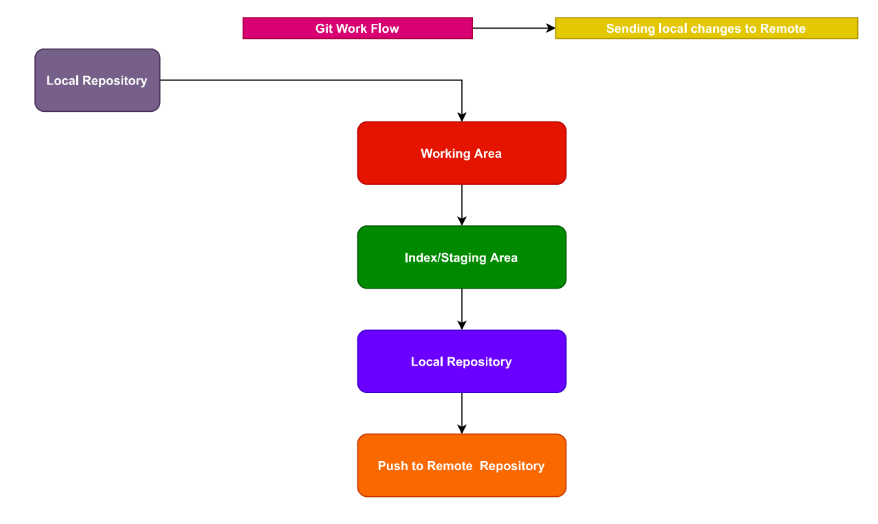
Remote repository: GitHub ==>> our created repository

Push: sending local repository changes to remote repository

Pull: sending the remote repository changes to the local repository. Pull request enables team mates to review and comment on the changes before merging to main branch, we also can see how many files are modified, we also can compare modified file with their old version.

Note: git follows ===>> 2 types of protocols ==>>> https and ssh.

Git Work Flow.



**Working Area:**

Any modifications we do to the local repository those modification is kept under working area.

**Staging/Index Area:**

This area is to stage the files we want to commit to local & remote repository

*git add README.md git*

*git add*

*git status*

This provides information about our local repository (working area, stating area & local repository)

**Commit:** When we do commit, it picks the files present in staging and commits to local repository.

*git commit -m “xxxxxxx”*

**Note:** Before we perform commit, git name and email must be configured

Initially, we need to clone remote repository to local.

Git Clone URL(Practical)

**Git clone URL** xxxxxxx

ls

2. Go to the inside **Local repository**.

cd xxx

3. files ==>>> create / modify ===>> **Working area.**

touch rk123

git status ==>> red colour ==>> **Working area.**

4. git add . / filename / \*

git status

file ==>> green colour ===>> **Indexing / staging area.**

5. we are getting the files to local repository ==>> **commit**

git commit -m 'xxxxx’ ==>> **Local repository**.

git log

40 digits commit ID ==>>> head ==>> head always points to latest commit.

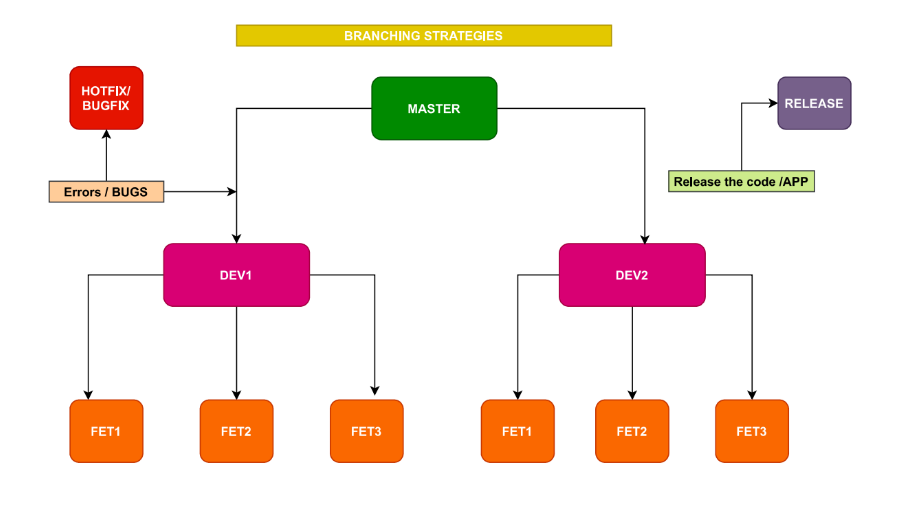
6. Push to **Remote repository**.

git push origin master

* push ==>>> sending local changes to remote repository.
* origin ==>> alias name ==>>> Github URL
* master ==>> default branch.

Git branching strategy

* In modern software development, **speed and agility are crucial when it comes to developing and releasing software**.
* However, when you have a large team of developers working simultaneously, branching and merging code can become messy fast.
* Therefore, teams need to have a process in place to implement multiple changes at once. This is where having an efficient branching strategy becomes a priority for these teams.
* Branches are primarily used as a means for teams to develop features giving them a separate workspace for their code. These branches are usually merged back to a master branch upon completion of work.
* In this way, features (and any bug and bug fixes) are kept apart from each other allowing you to fix mistakes more easily.
* This means that branches protect the mainline of code and any changes made to any given branch don’t affect other developers.



**Master branch:**

Master must contain well tested code application release happens from this branch by creating a release ‘tag’.

In real world no one directly work on master.

**Hotfixes:**

This branch is used for fixing production defects. Hotfixes branch is created from master

**Develop branch:**

This branch belongs to a specific team, code integration of this team members are done on this branch

Develop branch is created from master

**Feature branch:**

Belongs to a specific developer, where his feature in implemented, after completion of a feature changes are merged into his develop branch.

Feature branch is created from develop branch

**Release branch:**

This branch is to integrate changes done by multiple teams under them develop

A BRANCHING STRATEGY AIMS TO:

* Enhance productivity by ensuring proper coordination among developers
* Enable parallel development
* Help organize a series of planned, structured releases
* Map a clear path when making changes to software through to production
* Maintain a bug-free code where developers can quickly fix issues and get these changes back to production without disrupting the development workflow