# **Git Bash – Hub**

**Working Directory (local machine) 🡪 Staging Area (local machine)🡪 Local Repo (local machine)🡪 Remote repo (Server ...Git hub)**

**Git Life Cycle**

**It is the time span between git uploader and git receiver.**

**Git Uploader: - those who will responsible for creating Assignment on Git hub (cloud / server)**

**Ex :- Any senior person in company like Sr. developer, Sr. Tester, Manger , CEO , Client**

**Git- receiver: - Those who will provide Solution from that particular Assignment After downloading it from git server and perform solution activity on their local machine and after completing Assignment again push this solution to git- server**

**1. Stage First (Git- uploader) :-**

1. **Create new Project (Assignment)…. local m/c**
2. **Create local git –repo… local m/c**
3. **Add Project work to staging Area … local m/c**
4. **Commit this change to local repo… local m/c**
5. **Push this changes to master (Main) branch remote repo (Git hub ( git server))…. local m/c to git server**

**2. Stage Second (Git- uploader) :-**

1. **Assigning the task to respected person or Authority**

**3. Stage Third (Git- receiver) :-**

1. **Clone the assignment from the git server into local repo( local m/c)**
2. **Import project from local m/c into your work space**
3. **Do work over assignment/ project (add / modify features/ code /test cases)**
4. **Create new branch in remote repo (git server)**
5. **Add project to staging**
6. **Commit changes to local repo**
7. **Push the changes to remote repo via new branch**
8. **Request PR (pull request ) for review the code**

**4. Stage Fourth (Git- uploader) :-**

**Review code :-**

* **Case1 : Solution not satisfy the assignment , infom the respected person to it**
* **Case2: Solution satisfy the assignment, then provide Approval to merge code in to master branch in git hub (git-server)**

**\*GERNAL COMMAND :-**

1. **$ ls := show folder or file list available in Working directory**
2. **$ clear := clear the git bush work space**
3. **$ pwd := show the location of current working directory**
4. **$ git --version := show the current git version**
5. **$ cd folderName := enter the inside the project**
6. **$ git init := it will initialize(create) new empty repo in current directory(.git)(local repo)**
7. **$git status := show the status**
8. **$ git log := show how many commit commands done earlier (local Repo)**
9. **$ cat File name.File extension := show what is inside in file**

**\*BEFORE ADDING WE MUST SET CONFIGURATION WITH REPO AND IT CAN ONLY APPLICABLE FOR FIRST TIME**

1. **$ git config --global user.email "git hub mail id" := set git hub mail id with repo**
2. **$ git config --global user.name "git hub user name" := set git hub name with repo**

**\*ADD FILE FROM WORKING DIRECTORY TO STAGING AREA**

1. **$ git add -a := add all the files**
2. **$ git file name . file Extension := add specific file**
3. **$ git add . \* := if file extension is same then add all of them**

**\*ADD FILE FROM STAGING AREA TO LOCAL REPO**

**$ git commit -m " msg " := this will helps us move files from staging Area to local repo**

**\*AFTER ANY MODIFICATION IN THE FILES**

**Means consider if any file present in local repo , and you do some chances into it and you need to add local repo so that time you use below command**

1. **$ git commit -a -m " msg" := it will only work over modification files and add from work directory to stagging area and stagging area to local repo**

**\*HOW TO FIND DIFFERENCE IN FILE CONTENT BETWEEN**

**1. To see the difference in file content between working directory and staging area**

**$ git diff fileName.file extension = Show the difference between file content which present at Staging Area And Working Dictionary**

**ex:=**

**command .... $ git diff index.txt**

**diff --git a/index.text b/index.text**

**index fcb5845..b5bf6bc 100644**

**--- a/index.text**

**+++ b/index.text**

**@@ -1 +1,2 @@**

**animals**

**+birds**

**EXPALTION**

**a/index.text := represents source(stagging area)...To**

**b/index.text := represents destination/workspace (working dir)...from**

**fcb5845 := hash of file content from source/stagging**

**b5bf6bc := hash of file content from destination/workspace**

**100644:= git file mode**

**100 := represent the type of the file**

**644 := file permission ( rw(4+2)-r(4)-r(4) )**

**in linux command**

**4 := represnt read**

**2 := represnt write**

**1 := represnt excute**

**owner (hv permission to read and write)--->group(hv permission to only read)---> user(hv permission to only read)**

**(rw-r-r)**

**--- a/index.text := represents source file missing some lines(stagging)**

**+++ b/index.text := represents New lines added in destination file (working dir)**

**-1 := represents source file missing some lines(stagging)**

**+1,2 := represents New lines added in destination file (working dir) and total line is two**

**if anyline prefixed with space means it is uncahanged...ex := ( animals)**

**if anyline prefixed with + means it is added in destination(working directory) copy...ex := (+birds)**

**if anyline prefixed with - means it is removed from destination copy**

**2. To see the difference in file content between working directory and local repo( last committed or current commit)**

**command**

**$ git diff HEAD file name. file extension**

**HEAD := represent the last commit in local repo**

**-----------------------------------------------------------------------------------**

**3. To see the difference in file content between staging copy and local repo (last committed or current commit)**

**command**

**$ git diff --stagged HEAD file name. file extension**

**--stagged := represent the staging area**

**-----------------------------------------------------------------------------------------------------**

**4.To see the difference in file content between working directory and local repo(Specific commit )**

**for that get commit id local repo (Specific commit )**

**$ git log --oneline**

**command**

**$ git diff commit id file name. file extension**

**--------------------------------------------------------------------------------**

**5.To see the difference in file content between staging copy and local repo(Specific commit )**

**for that get commit id**

**$ git log --one line**

**command**

**$ git diff --stagged commit id**

**-------------------------------------------------------------------------------------------------------**

**5.To see the difference in file content between local repo(Specific commit )**

**for that get commit id**

**$ git log --one line**

**Command**

**$ git diff commit id1 commit id2 file name. file extension**

**-----------------------------------------------------------------------------------------------------------**

**GIT RM**

**Used for remove file form working dir and staging area**

**$ git ls -files := show the staging area files**

**----------------------------------**

**1. Remove Files From Both Staging & Working Directory**

**$ git rm filename.fileExtension := remove mentioned file from staging and working directory**

**$ git rm -r. := remove all files from staging and working directory**

**-----------------------------------------------------------------------**

**2.Remove Files Only From Staging**

**$git rm --cached filename.fileExtension := remove mentioned file only from staging area**

**------------------------------------------------------**

**3. Remove Files Only From Working Directory**

**$ rm filename.fileExtension := remove mentioned file from only working directory**

**-------------------------------------------------------------------------**

**CHECKOUT COMMANDS**

**Checkout command used for discard (delete) un-staged changes in the tracked files of working directory**

**(checkout command used for delete file content at working dir which is not update in staging or local repo )**

**1. checkout command applicable for only tracked files not for untracked files**

**2. only for working directory**

**3. To discard un-staged changes ( the changes which are not added to staging area)**

**4. in the tracked files (the files which are already added to staging area/ commit)**

**Un-staged changes := means one file added to staging and local repo but we do something changes(something adding to the file) at working dir and we are not updated it to staging and local repo this situation known as unstagged changes.**

**tracked files := those files present in working dir ,staging area and local repo**

**Untracked files:= those files present in working dir but not in staging area and local repo**

**---------------------**

**$ cat FileName.FileExtension := show what is inside in file**

**$ git checkout -- Filename.FileExtension := delete un-staged changes from file at working dir**

**GIT RESET COMMAND**

**it used to remove the changes made file content**

**$git reset --mode commit id := it will remove file**

**note := what ever commit id you passed, above all commit will be remove**

**it will not help us to remove specific id**

**ex := commit3,commit2,commit1**

**if you pass commit id for commit1 then commit 2 and 3 will be remove**

**MODE:=**

**1. mixed ( $git reset --mixed commit id OR $git reset --commit id )**

**it is the default mode**

**to discard commit in the local repository and to discard changes in staging area**

**it will not touch working directory.**

**2. soft ( $git reset --soft commit id )**

**to discard commit in the local repository only**

**it will not touch staging area and working directory.**

**3. hard ( $git reset --hard commit id )**

**to discard file from all areas (working + staging + local repo)**

**-------------------------------------------------------------------------------------------------------------------------------------------**

**GIT BRANCHING**

**concept used for parallel code development .**

**maintained clean code**

**MASTER**

**is main branch in git or default branch.**

**CUSTOMISE BRANCH**

**1. while creating any new branch, at that time whatever code or file available in Master branch, this all data inherited into this particular branch(till local repo)**

**2. After creating new branch it is totally independed from other branches(means it will doest not affect any branch )**

**3. After completing work in branch there is two ways to summit, first to merge with the master branch and second push it individually in repo**

**4. (\*):= active branch**

**COMMANDS**

**$ touch filename.fileExtension := create new file in working dir**

**$ git branch OR git status := show the available branches**

**$ git branch branch name := create mentioned new branch**

**$ git checkout branch name := Switched to mentioned branch**

**$ git checkout -b branch name:= creating and Switched to new branch.**

**-----------------------------------------------------------------------------------------------------------------------**

**MERGE**

**it is concept regarding to branching**

**After creating and completing work in branching we have options, like one is merge to the master branch and second one is push individually in repo (Remote repo)**

**1. MARGE TO THE MASTER BRANCH**

**To merge any branch, for that first we need to switch to the master branch and execute below command**

**( $ git checkout branch name)**

**$ git merge name of the branch(which you want to merge)**

**Types**

**1. Fast-Forward merge:=**

**means after creating new branch, there is no any changes happed in master branch**

**there is no chances of conflict**

**after merging into master branch, new head will be last commit in merged branch(feature branch)**

**2. three-way marge:=**

**means after creating new branch, there is some kind of changes made in master branch**

**there is chances of conflict , if there is need to resolved conflict manually**

**after merging into master branch, new head will be merge commit created by git (automatically) [:wq! = save and exit]**

**Resolve Conflict And Marge :=**

**after receving error msg like conflict ...so first reslove Conflict then Marge**

**,either delete one of them or keep both modifition togther and again do commit**

**$vim FileName.FileExtension := inser into file**

**--insert-- := linux edit option enable**

**:wq! := Save and Exit**

**dd := Delete line**

**(After resolving conflict mc head will be created ,so head must be add into stagging and local repo (dop commit over mc head))**

**$ git branch -d baranchName := delete mentioned branch**

**-------------------------------------------**

**Importing From Remote Repo ( Git Hub )**

**1. clone = command used where we don’t have local repo (At beginning of project)**

**$ git clone "repo url"**

**2. push = push the local repo in to remote repo ( Git hub)**

**$ git push –u origin branch name (main / master)**

**3. pull =command used where we have already local repo and do some modification which available in git repo**

**$ git pull –u origin branch name (main/ master)**