**GIT Commands**

|  |  |
| --- | --- |
| GIT status | If current branch has any untracked file |
| GIT log | If current branch has any Commit IDs |
| GIT log --oneline | Commits IDs in one line |
| Git log ---pretty =format:”%h -%an -%ar - %s” | Logs in better format |
| Git log –stat |  |
| GIT add . | Staging the file<moving the file from local to staging. Red to green> |
| GIT commit –m <message> | Moving file from staging to local repository |
| GIT reset HEAD <file name> | To move file back from staging to unstage<green too red> |
| GIT config --global help.autocorrect 1 | Auto corrects the spelling in the command if any  **Note**: global 🡪user level; --system🡪for whole system; If not anything, then it’s at the Project level |
| GIT config --global user.name “AtulS” | For tracking your changes <This should be same as GITHUB> |
| GIT config --global user.email “xyz.gmail.com” | For tracking your changes<This should be same as GITHUB> |
| Rm <filename> | This will remove the file, but you need to stage and then commit the changes |
| Git rm <filename> | File gets to staged and then you need to commit |
| Git rm - -cached <File name> | If you want to untrack a file after it has been committed  Using this command, it will show both in green and red  You need to use commit command to remove green color, but red remains as Git is not tracking  File is physically present |
| Git ignore <file/folder> | List by file names or folder if you don’t need as a part of source code. Removing junk files. Usually new files before staging and committing   1. Create a gitignore file<vi .gitignore> 2. Place all the file names with or without wild cards “\*” 3. This will not be tracked by status command 4. Untracked file is not the part of your deployment |
| GIT stash  After this command git status will show working directory is clean  When again we need these files, we can recover them  GIT stash list  Stash@{0} ->Always recent  Git stash show stash@{0}  To recover  Git stash apply stash@{0} | 1. Remove files which are staged and not committed and also at the same time you don’t want to remove the file 2. You have staged the file[in Green], but manager said we need to remove these file and will include in the new sprint 3. Git has some reserved space[Stash], where you can save in git memory which you don’t need currently 4. Get staged file back and commit using git stash apply command |
| GIT tag  GIT tag –a v1.2.23.1 –m “This is my HOT fix” <Assign the specific commit id at the last> In yellow is a tag. | 1. When done for release/minor release /Hot fix, then assign a tag to it 2. When no commit id is specified, it takes by default the recent Commit IDs. Tag is assigned to recent commit ID 3. Tags are for versioning 4. Tags and commit id goes hand in hand 5. Tag is particular state of the file |
| GIT show<tag name>/<commit id> | You can see particular changes for a particular tag or commit id  So assign tag to the commit id |
| Head🡪master | Head always points to the recent commit ids |
| Git revert |  |

**Remote queries**

|  |  |
| --- | --- |
| Git remote –v | This will list out the GIT hub urls linked to this local repo |
| Git remote add origin <github url> | Connecting to the github for the first time |
| Git push –u origin master | To push code to GITHUB master branch <username will be asked> |
| **Release tab in GITHUB**  Git push --tags | Push command only pushes the data not tags  Tags need to be pushed separately  You can download the zip file from github now |
| Git pull | This will download the changed code/file only from github /updates |
| Git fetch | Shows you the changes, but does not update your data in the local |
| Scenario: Removing a file from local | Add and commit and push |
| Diff between pull and fetch | Mentioned as above |
| Merge Conflict(\*\*\*Imp) | Changes in the same file at Local and hub, when you do pull, on git pull both of them will be displayed. Now discuss which changes to keep, make changes manually and commit and push.  This will show changed data in the hub that was pushed from local |
| Git pull request concept (\*\*\*Imp) | The process of reviewing and approving the changes, before merging to the main branch   1. Copying the repository from a different account to your account 2. Make changes🡪Commit, in the file (in your account) 3. Now, to update the changes to Original account. Here comes pull request concept 4. There is a “New pull request” button. Click and it goes to the base repository 5. Under the Pull request tab, you will see the request,, just click “Merge Pull Request”   The change will be applied to the original project |
| Fork concept  Diff between clone and fork | 1. Copying the repository from a different account to your account |
| Git checkout master  Git merge <branch name> | 2 cases:   1. Master does not have any new commit ID(fast –forward) 2. Master has new commit id already: new commit id created after merge along with existing commit ids |
| Git clone <git hub url> | Copying and downloading the repository to local machine  When doing for the 1st time |
| Git branch  Git branch <branch name>  Git branch checkout | List out all branches  Creating a new branch  Switching to another branch |
| Git branch -d <Branch name> | Deleting a branch when merged or there is no need |
| Fast-forward concept |  |
| Git rebase |  |
| Git reset |  |
| Git cherry –pick <commit id> | Selecting only 1 commit id. Only want to pick a specific commit from a new branch and leave others |
| Git diff –summary <commit id1> <commit id 2> |  |
| Git gc --prune | Garbage collection |