**GIT**

**Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

**Git is currently the most popular version control system.**

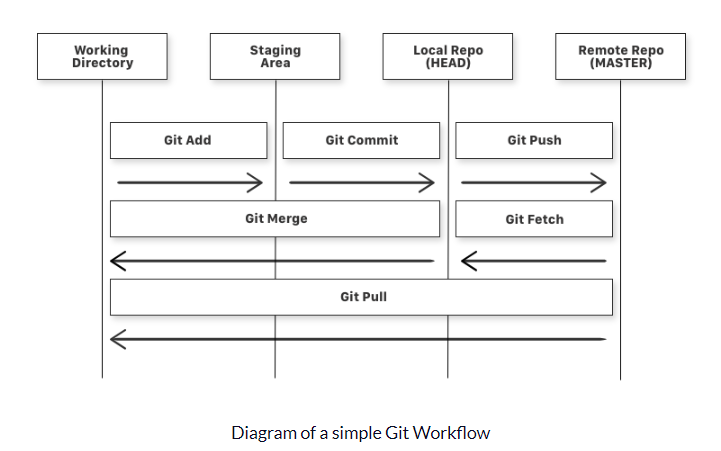
**This is, among others, because it is fast, distributed, branched, and free.**

**The very idea of working with Git is fairly easy, but the number of commands is quite large, and quite a few of them are rarely used.**

**Git commands can be divided into several types, depending on what they are for.**

**These can be configuration commands (e.g. config), to make changes to the repository (e.g. commit, push), branching and Merging (e.g. checkout, merge) or inspection and comparison (e.g. log, diff).**

**Workflow of Git :**



**Git commands types and examples**

1. **init Creates a new local (empty) repository with the default branch (master), but without any git commit.**

**$git init**

1. **Clone Creates a local copy of the remote repository.**

**$git clone <repo address> <local directory>**

1. **Bundle Create a single archive file with all refs needed to restore the repository.**

**$git bundle create <archive name> –all**

1. **add Add changes to the Staging Area.**

**$git add <file\_name>**

1. **Commit Commit changes from the Staging Area.**

**$git commit -m “Message”**

1. **Push Sends changes to a remote repository (origin) to a given branch.**

**$git push origin <branch\_name>**

1. **Fetch Take changes from the external repo but do not include them in local branches.**

**$git fetch origin**

1. **Pull It downloads and immediately integrates changes to the local branch.**

**Pull = fetch + merge**

**$git pull origin <branch\_name>**

**Above call is equal to this two below:**

**$git fetch origin <branch\_name>**

**$git merge origin/<branch\_name>**

1. **reset Reset current HEAD to the specified state (e.g. commit):**

**–soft differences will be preserved in the Staging Area,**

**–mixed the differences will be kept in the Working Directory,**

**this is the default scope,**

**–hard the differences will be completely removed.**

**$git reset –soft <file\_name>**

**$git reset –hard**

1. **Revert Rolls back the changes by creating a new commit that is the exact opposite of the one being reverted. Safe operation – will not spoil the commit history**

**reverts the last commit.**

**$git revert <commit\_SHA>**

**$git revert HEAD~1**

1. **Branch Basic operations on branches.**

**– showing the list of all branches,**

**– creating a new branch,**

**– deleting the specific branch.**

**$git branch**

**$git branch <branch\_name>**

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

1. **Checkout Switches to the given branch.**

**Creates a new branch and switches to it immediately.**

**$git checkout <branch\_name>**

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

1. **Remote Manage a set of tracked repositories.**

**Set remote server address (e.g. after init operation).**

**Displays a list of currently set connections.**

**$git remote add origin <repository URL address>**

**$git remote -v**

1. **Merge Concatenates changes from the given branch to the currently active branch. Possible merge conflicts.**

**$git merge <branch name>**

1. **Rebase Reapply commits on top of the current and updated branch.**

**The current branch is re-based.**

**Flatten the history, avoiding merge-commits.**

**$git rebase <branch\_name>**

1. **Status Shows the current status of the Working Directory and Staging Area.**

**$git status**

1. **Log It shows the commit list with some details depending on the parameters provided.**

**Prints log for the last 3 commits.**

**Prints log for author’s commits, not older than 1 week.**

**$git log -3**

**$git log –author=”babu” –after=”1 week ago”**

1. **diff Show changes between two items:**

**– commits difference**

**– branches difference**

**$git diff**

**$git diff <SHA\_commit1> <SHA\_commit2>**

**$git diff <branch1\_name> <branch2\_name>**

1. **Config Allows for basic configuration, such as username, email address or default text editor, etc.**

**Deletion of a given config.**

**Be aware that there are 3 levels of configuration: local, global, system.**

**$git config user.name “babu”**

**git config –local user.email “babu\_devops@gmail.com”**

**git config –unset <option name>**

1. **configuration: global**

**# Setting Up User Name and Email Address**

**$ git config --global user.name "babu"**

**$ git config --global user.email** [babu\_devops@gmail.com](mailto:babu_devops@gmail.com)

<https://www.freecodecamp.org/news/learn-the-basics-of-git-in-under-10-minutes-da548267cc91/>

**Using merge/rebase command to merge the branches**

**Merge demo**

**Git merge is used integrate changes from one branch into another branch.**

**Create a project mergedemo and create master and dev branches**

**# git init**

**# touch m1.txt**

**# git status**

**Added to staging and commit**

**# git add . && git commit -m "Added m1.txt file"**

**# git log**

**Create and checkout to dev branch**

**# git checkout -b dev**

**# git log**

**Create d1.txt file dev branch**

**#touch d1.txt**

**#git add . && git commit -m "added d1.txt in dev branch"**

**Checkout to master and create m2.txt file**

**# git checkout master**

**# touch m2.txt**

**# git add . && git commit -m "added m2.txt file"**

**# git log**

**Checkout to dev and merge to master**

**# git checkout dev**

**# git merge master**

**#git log**

**Reverting the merge**

**# git reset 64099d2**

**# git log**

**If we are trying to checkout master branch will get error need to delete the working area files.**

**# git checkout master (It will get error)**

**# git clean –n**

**# git clean –f**

**# git checkout master (It will work)**

**Verify the logs in master and dev branch.**

**# git log**

**# git log**

**Rebase demo**

**Git rebase is used integrate changes from one branch into another branch.**

**Create a project rebasedemo and create m1.txt file in master branch.**

**# mkdir rebasedemo**

**# touch m1.txt**

**# git init**

**# git add . && git commit -m "added m1.txt file"**

**# git log**

**Create dev branch and add d1.txt file**

**# git checkout -b dev**

**# touch d1.txt**

**# git add . && git commit -m "added d1.txt file"**

**# git log**

**Create m2.txt in master and merge to the dev branch using rebase process**

**# git checkout master**

**# touch m2.txt**

**# git add . && git commit -m "added m2.txt file”**

**# git log**

**# git checkout dev**

**# git rebase master**

**# git log**

**git merge Vs rebase**

**Git rebase and merge both integrate changes from one branch into another**

**git merge**

**- Is a non-destructive operation.**

**- Existing branches are not changed in any way.**

**- Creates a new merge commit in the feature branch.**

**git rebase**

**- Moves the entire feature branch to begin on the tip of the master branch.**

**- Re-writes the project history.**

**- We get much cleaner and linear project history.**

**Note: Use merge if we have any issue we can reset easily**

**Git branching strategy:**

**Branching strategies coordinate work to allow for easier integration of changes and releases.**

**The point of a branching strategy is to efficiently manage code changes.**

**This workflow will impact both**

**Developer and deployment workflows. It’s mainly divided into 2-categories.**

**1. Main branches**

**2. Supporting/Supplementary branches**

**Main branches :**

**The development model is greatly inspired by existing models out there.**

**The central repo holds two main branches with an infinite lifetime.**

**- master branch**

**- develop branch**

**Supporting/Supplementary branches**

**Supporting branches to aid parallel development between team members, ease tracking of**

**features, prepare for production releases and to assist in quickly fixing live production problems.**

**Unlike the main branches, these branches always have a limited life time, since they will be**

**removed eventually.**

**- feature branch**

**- release branch**

**- bugfix branch**

**- hotfix branch**

**Master branch:**

**Master must contain well tested code, application release happens from this branch by creating a release ‘tag’. All development code is merged into master in sometime. In real world no one directly work on 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. This branch contains pre-production code. When the features are finished then they are merged into develop branch.**

**Feature branch:**

**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/UAT branch:**

**This branch is to integrate changes done by multiple teams under their develop branch. Release branches support preparation of a new production release. They allow many minor bug to be fixed and preparation of meta-data for a release. May branch off from develop and must merge into master and develop.**

**Bugfix branch:**

**This branch is used for fixing UAT defects. Bugfixes branch is created from release.**

**Bugfix branches are necessary to act immediately upon an undesired status of release. May branch off from release and must merge into release branch.**

**Hotfix branch:**

**This branch is used for fixing production defects. Hotfixes branch is created from master. Hotfix branches are necessary to act immediately upon an undesired status of master. May branch off from master and must merge into master and develop**

**Git clone**

**In Git, cloning is the act of making a copy of any target repository. The target repository can be remote or local. You can clone your repository from the remote repository to create a local copy on your system. Also, you can sync between the two locations.**

**GIT Fetch**

**Git "fetch" Downloads commits, objects and refs from another repository. It fetches branches and tags from one or more repositories. It holds repositories along with the objects that are necessary to complete their histories to keep updated remote-tracking branches**