**Git:**

In building any project we used to maintain versions while we are adding features to it. Before Git, we used to maintain many directories for different versions. For every new version we need to copy entire directory and we need to add new features to it. As this process is very tedious, we may be lost the sequence of the versions.

In Git, it is very easy to maintain the different version and it well suits for the collaborative working environment. Git maintains every version itself when we do commit on the directory.

**Installing Git:**

Download the git file from  **<https://git-scm.com/download/win>** and execute. Set the path in the Environment PATH Variable.

Type git - -version in the command prompt. It will print git version.

**Setting Repository:**

1) Initialize the repository

Command:

**Git init repositoryName**

2) Add all the files to the repository

Command:

**Git add .** (To add all the files)

**Git add filename**(To add the one file)

3) Save the changes

**Git commit**

**Git commit:**

When we commit on the git, It saves the current state of our directory. Git maintains every snapshot of our directory in a logical way. It won’t maintain an entire copy of the directory, It maintains only maintains the changes from the previous commit to present commit.

**Command:**

**Git commit –m “any message”**

**Pushing and pulling to between localmachine and remote machine:**

Add URL of the remote machine.

**Git remote add origin URL**

Push the repository to remote machine

**Git push origin master**

Pull the repository from remote machine

**Git pull origin master**

**Git Branches:**

We can maintain many branches on its parent. By maintaining braches we can work individually and later we can merge those branches.

**Commands:**

**Git branch NewBranch**

**Git checkout NewBranch; git commit**

**Git Merge:**

After making branches we can merge branches which have the same parent.

**Commands:**

**Git branch NewBranch**

**Git checkout –b NewBranch**

**Git checkout master**

**Git commit**

**Git merge NewBranch**

**Git Rebase:**

This is used to make a linear sequence of commits.

**Commands:**

**Git branch NewBranch**

**Git checkout NewBranch; Git commit**

**Git checkout master**

**Git commit**

**Git checkout bugFix**

**Git rebase master**

**Head:**

Head always points to currently checkout commit.

**Git checkout c2**

**Relative References**:

Git uses a hashtag for particular commit. Specifying commit by their hash is a tedious thing, So we use relative refs.

Moving upwards one commit at a time with ^

**Git checkout HEAD^**

Moving upwards a number of times with ~<num>

**Git checkout HEAD~4**

**Branch Forcing:**

With this, we can move the master branch

**Git branch –f master HEAD~3**

**Reversing The changes on Git:**

**Git Reset:** Git reset reverts changes by moving a branch reference backwards in the time to an older commit.

**Git reset HEAD~1**

**Git Reverting:**

With reverting, we can push out our changes to share with others.

**Git revert HEAD**

**GIt Cherry-pick:**

We can copy series of commits below our current location.

**Git cherry-pick C2 C4**

**Advantages:**

1) Git won’t duplicate the data, Instead, it only maintains changes in a logical way.

2) Git is simple to use if we understand the concept of it.

3) Branching is the one best thing, Using this we can divide the work later we can merge the work into master. It suits well for the collaborative environment

4) The data will not be lost. We can easily revert back to the previous version.

5) It uses the disk efficiently.

6) We can work on some one’s repository and modify the source code in a separate branch and we drop pull request to them.

**Disadvantages:**

1. There will be many commands which we need to remember while using it.
2. We should have some knowledge of git before using git.