**Training Program: Module 0**

**Git Installation**

First of all, I installed Git with Apt after that Git setup is done by using git config command. For this, name and email address is provided because git embeds this information into each commit we do.

**Git Tutorial**

I learned and exercise the following Git commands.

|  |  |
| --- | --- |
| **Git Basic Commands** | |
| Command | Description |
| *git init <directory>* | It creates empty Git repository in specified directory. If no directory is specified then it initializes current directory as a git repository. |
| *git clone <repo>* | Used to create local copy of remote repository located at <repo> on your computer. |
| *git config user.name <name>* | Set author name that to be used for all commits in current repository. |
| *git add <directory>* | Stage all changes in directory for the next commit. And replace the <directory> with <file> to stage a specific file. |
| *git commit –m “<message>"* | Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message. |
| *git status* | List which files are staged, unstaged, and untracked. |
| *git log* | Display the entire commit history using the default format. |
| *git diff* | Show unstaged changes between your index and working directory or between two files on disk. |
| **Undoing Changes** | |
| *git revert <commit>* | Create new commit that undoes all of the changes made in <commit>, then apply it to the current branch. |
| *git reset <file>* | Remove <file> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes |
| *git clean -n* | Shows which files would be removed from working directory. Use the -f flag in place of the -n flag to execute the clean |
| **Git Config** | |
| *git config --global user.name <name>* | Set the author name that to be used for all commits by the current user. |
| *git config --global user.email <email>* | Set the author email that to be used for all commits by the current user. |
| *git config --global alias. <alias-name> <git-command>* | Create shortcut for a Git command. |
| *git config --system core.editor <editor>* | Set text editor used by commands for all users on the machine. <editor> arg should be the command that launches the desired editor (e.g., vi). |
| *Git config –global --edit* | Open the global configuration file in a text editor for manual editing. |
| **Git Log** | |
| *git log -<limit>* | Limit number of commits by <limit>. E.g. git log -5 will limit to 5 commits. |
| *git log --oneline* | Condense each commit to a single line. |
| *git log --stat* | Include which files were altered and the relative number of lines that were added or deleted from each of them. |
| *git log --author= ”<pattern>”* | Search for commits by a particular author. |
| *git log --grep=”<pattern>”* | Search for commits with a commit message that matches <pattern> |
| *git log -p* | Display the full diff of each commit |
| *git log <since>..<until>* | Show commits that occur between <since> and <until>. Args can be a commit ID, branch name, HEAD, or any other kind of revision reference. |
| *git log -- <file>* | Only display commits that have the specified file. |
| *git log --graph --decorate* | -graph flag draws a text based graph of commits on left side of commit msgs. --decorate adds names of branches or tags of commits shown. |
| **Rewriting Git History** | |
| *git commit --amend* | Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit’s message |
| *git rebase <base>* | Rebase the current branch onto <base>. <base> can be a commit ID a branch name, a tag, or a relative reference to HEAD. |
| *git reflog* | Show a log of changes to the local repository’s HEAD. Add --relative-date flag to show date info or --all to show all refs. |
| **Git Branches** | |
| *git branch* | List all of the branches in your repo. Add a <branch> argument to create a new branch with the name <branch>. |
| *git checkout -b <branch>* | Create and check out a new branch named <branch>. Drop the -b flag to checkout an existing branch. |
| *git merge <branch>* | Merge <branch> into the current branch. |
| **Remote Repositories** | |
| *git remote add <name> <url>* | Create a new connection to a remote repo. After adding a remote, you can use <name> as a shortcut for <url> in other commands |
| *git fetch <remote> <branch>* | Fetches a specific <branch>, from the repo. Leave off <branch> to fetch all remote refs. |
| *git pull <remote>* | Fetch the specified remote’s copy of current branch and immediately merge it into the local copy. |
| *git push <remote> <branch>* | Push the branch to <remote>, along with necessary commits and objects. Creates named branch in the remote repo if it doesn’t exist. |
| **Git Diff** | |
| *git diff HEAD* | Show difference between working directory and last commit. |
| *git diff --cached* | Show difference between staged changes and last commit |
| **Git Reset** | |
| *git reset* | Reset staging area to match most recent commit, but leave the working directory unchanged |
| *git reset --hard* | Reset staging area and working directory to match most recent commit and overwrites all changes in the working directory |
| *git reset <commit>* | Move the current branch tip backward to <commit>, reset the staging area to match, but leave the working directory alone. |
| *git reset --hard <commit>* | Same as previous, but resets both the staging area & working directory to match. Deletes uncommitted changes, and all commits after <commit> |
| **Git Rebase** | |
| *git rebase -i <base>* | Interactively rebase current branch onto <base>. Launches editor to enter commands for how each commit will be transferred to the new base. |
| **Git Pull** | |
| *git pull --rebase <remote>* | Fetch the remote’s copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches. |
| **Git Push** | |
| *git push <remote> --force* | Forces the git push even if it results in a non-fast-forward merge. Do not use the --force flag unless you’re absolutely sure you know what you’re doing. |
| *git push <remote> --all* | Push all of your local branches to the specified remote. |
| *git push <remote> --tags* | Tags aren’t automatically pushed when you push a branch or use the --all flag. The --tags flag sends all of your local tags to the remote repo. |
| **Git Cherry-Pick** | |
| *git cherry-pick* | Apply the changes introduced by some existing commits. Given one or more existing commits, apply the change each one introduces, recording a new commit for each. This requires your working tree to be clean (no modifications from the HEAD commit). |
| *git cherry-pick --edit* | With this option, git cherry-pick will let you edit the commit message prior to committing. |
| *git cherry-pick master* | Apply the change introduced by the commit at the tip of the master branch and create a new commit with this change. |
| *git cherry-pick ..master* | Apply the changes introduced by all commits that are ancestors of master but not of HEAD to produce new commits |
| *git cherry-pick master~4 master~2* | Apply the changes introduced by the fifth and third last commits pointed to by master and create 2 new commits with these changes. |

**Summary**

After this, I am able to do all the basic local Git operations – creating or cloning a repository, making changes, staging and committing those changes, and viewing the history of all the changes the repository has been through.