**GIT Assignment 1 -**

**VCS Into**

Version control systems are a category of software tools that help a software team manage changes to source code over time.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made,

developers can turn back the clock and compare earlier versions of the code to help fix the mistake

while minimizing disruption to all team members.

**Phylosophy & Concept**

The whole concept of making VCS is to ensure that the develops will feel safe hwne they are working on the project because traditionally there is problem we need to make whole software then only we can check the bugs now here what we are doing changes are occurring everyday testing everyday and also your data is safe in you system as well as in remote location also.

It gives flexibility and also saving time

The most important thing is we can switch to our previous version/state if any problem occurs.

**Distributed VS Centralized**

The main difference between the two classes is that Centralized VCSs keep the history of changes on a central

server from which everyone requests the latest version of the work and pushes the latest changes to.

This means that everyone sharing the server also shares everyone’s work. Sourceforge.net uses this type of versioning in their projects.

On the other hand, on a Distributed VCS, everyone has a local copy of the entire work’s history.

This means that it is not necessary to be online to change revisions or add changes to the work.

“Distributed” comes from the fact that there isn’t a central entity in charge of the work’s history,

so that anyone can sync with any other team member. This helps avoid failure due to a crash of the central versioning server.

Open source projects, such as Mozilla Firefox, tend to use this type of versioning

**Working Copy**

It is the actual work you di on your work space and you maintain the copy of that work that is called working copy

**Staging Area**

After making changes in files or code you need to add those changes in stagging area ( which is also known as indexing area )

before any commit in local repo.

You can even remove those changes which you add in stagging area before commit.

**Local Repository**

This is the repo where we commit all our changes before pushing it to Remote repo.

This repo conating all you work history.

**Remote repo**

This is the repo where all the team member or developer actually shring there work , pushing there changes in this repo and

pulling all the changes from this repo so

that they can maitain the work history in local repo to know what is going on there project.

**Remote Name**

It is just the name we are using for our remote repo

We can rename also by

Git remote rename origin destination

It will change the name from origin to destination

We can check our remote name by

Git remote -v

**Change History/Log**

The changes which we made in our local repo we can see those changes by typing git log.

**git init ( to make that particular directory your local repo )**

**git add ( to add your changes in stagging area )**

**git clone <repo> <directory> ( to create a clone of your remote repo / bare repo )**

**git status ( to check the status of your local repo if any untracked file is there )**

**git diff ( will tell you the difference between you previous and existing file )**

**git commit -m "message" <filename > ( to commit your changes to local repo )**

**git log ( it will tell you the working history of you local repo 0**

**git rm --cached file1.txt ( it will remove the file from git repo not delete it from the local file system )**

**git mv oldname newname ( it will remove the file oldname and copy the data of oldname into new name )**

**git branch <branchname> ( it will create the new branch independent of master branch so that we can do parallel development)**

**git checkout ( to logout from particulatr branch )**

**git fetch ( it will fetch the version you want without merging them into your working directory so that you can view and then merge.**

**git pull ( it will pull the version which you want and merge them into you current working directory)**

**we can say that - git pull = git fetch + git merge**

**Branching**

It is nothing but the way to do parallel development.

The default branch name in Git is master. As you initially make commits, you're given a master branch that points to the last commit you made.

Every time you commit, it moves forward automatically.

Suppose you have created a new branch and working on that particular branch then pointer ( HEAD ) move to that particular branch

on which you are working.

**Merging**

When we crated one branch other than master and you are working on both parallel now you are confimr that the work you did

on other branch is ok and you want to merge it with master.

Then we use git merge remeber you need to run this command in the same branch where you want to merege all the changes.

**Resolve Merge conflict**

Whe we are working on 2 different branches and at certain point we need to merge those branches but the problem is both branch having one

file same and in the same line so when yu try to merge these

branches it will ask you that which changes you wnat to merge and which changes you want to remove.

**HEAD**

It is nothing but a pointer which will tell you about your working branch.

The default branch name in Git is master. As you initially make commits, you're given a master branch that points to the last commit you made.

Every time you commit, it moves forward automatically.

**Practical Assignment**

**git init ( Initialize a directory as git repository0**

**Create a blank files like a.txt, b.txt.**

**Write "Hi Team Ninja" into a.txt**

**Write "This is VCS:Git" into b.txt**

**Set git config variables**

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

**git config --global email.name "email"**

**git add - A ( stagging )**

**git commit -a -m " ..." ( commit)**

**git branch ninja ( creation of branch)**

**git checkout ninja ( switch to bracnh ninja)**

**Update file b.txt with ""This is VCS:Git. This is ninja branch"**

**git diff <tag(or)branch name> ( will compare each modified files in your file system against the files in the specified branch)**

**git diff <tag1(or)branch1 name> <tag2(or)branch2 name>: will compare all modified files between two branches / tags**

**git diff --name-only : Show only names of changed files, not the contents.**

**git diff --name-status : Show only names and status of changed files.**

**git diff --cached (or --staged) : compares only the files which are staged/indexed**

**Commit your changes in ninja branch.**

**root@ubuntu-xenial:/devops# git diff HEAD~**

**diff --git a/jenkins b/jenkins**

**new file mode 160000**

**index 0000000..5380d03**

**--- /dev/null**

**+++ b/jenkins**

**@@ -0,0 +1 @@**

**+Subproject commit 5380d03729fa95fe86b6c7ac62a2dda6b958d9e2**

**git mv b.txt c.txt ( Rename your file b.txt to c.txt )**

**Commit your changes.**

**Remove file c.txt.**

**Commit your changes.**

**Create a file text.txt and add it.**

**Remove it from the staging area ( git reset <file> / git reset HEAD )**

**Create a account on github.com**

**Fork ot-training/jenkins repo into your account.**

**Clone your repo into your machine. ( git clone <url> )**

**Clone your repo into your machine.**

**Add a file to the attendees directory that describes you (ex. filename wouild be your name like saurabh.vajpayee.txt).**

**Commit your changes. Write a meaningful message like "Added information about YOUR NAME HERE."**

**Save your changes into remote repo.**

**Check list of changes in your repo.**

**Add your name into attendees/assignments/day1/attendees.md**