EXPERIMENT NO. 2

SEMESTER:VI DATE OF PERFORMANCE:

SUBJECT: MlOps

DATE OF SUBMISSION:

NAME OF THE STUDENT: ROLL NO.:

Aim:
Setting up a Version Control System (VCS) for ML Projects:
a. Experiment with popular VCS tools like Git and create a repository for ML projects.
b. Learn to track code changes, collaborate with team members, and manage different branches.

Theory:

What is a "version control system"?

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

Why Version Control system is so Important?

As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes to some specific kind of functionality/features. So in order to contribute to the product, they made modifications to the source code(either by adding or removing). A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made. A separate branch is created for every contributor who made the changes and the changes aren't merged into the original source code unless all are analyzed as soon as the changes are green signaled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

Basically Version control system keeps track on changes made on a particular software and take a snapshot of every modification. Let's suppose if a team of developer add some new functionalities in an application and the updated version is not working properly so as the version control system keeps track of our work so with the help of version control system we can omit the new changes and continue with the previous version.

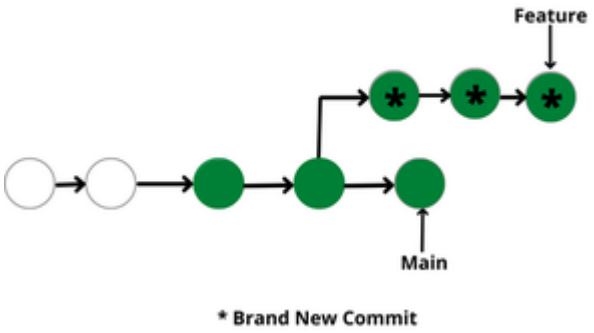
Benefits of the version control system:

- Enhances the project development speed by providing efficient collaboration,

- Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,
- Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
- Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this VCS,
- For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is Git, Helix core, Microsoft TFS,
- Helps in recovery in case of any disaster or contingent situation,
- Informs us about Who, What, When, Why changes have been made.

Use of Version Control System:

- A repository: It can be thought of as a database of changes. It contains all the edits and historical versions (snapshots) of the project.
- Copy of Work (sometimes called as checkout): It is the personal copy of all the files in a project. You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.
- Working in a group: Consider yourself working in a company where you are asked to work on some live project. You can't change the main code as it is in production, and any change may cause inconvenience to the user, also you are working in a team so you need to collaborate with your team to and adapt their changes. Version control helps you with the, merging different requests to main repository without making any undesirable changes. You may test the functionalities without putting it live, and you don't need to download and set up each time, just pull the changes and do the changes, test it and merge it back. It may be visualized as.



Types of Version Control Systems:

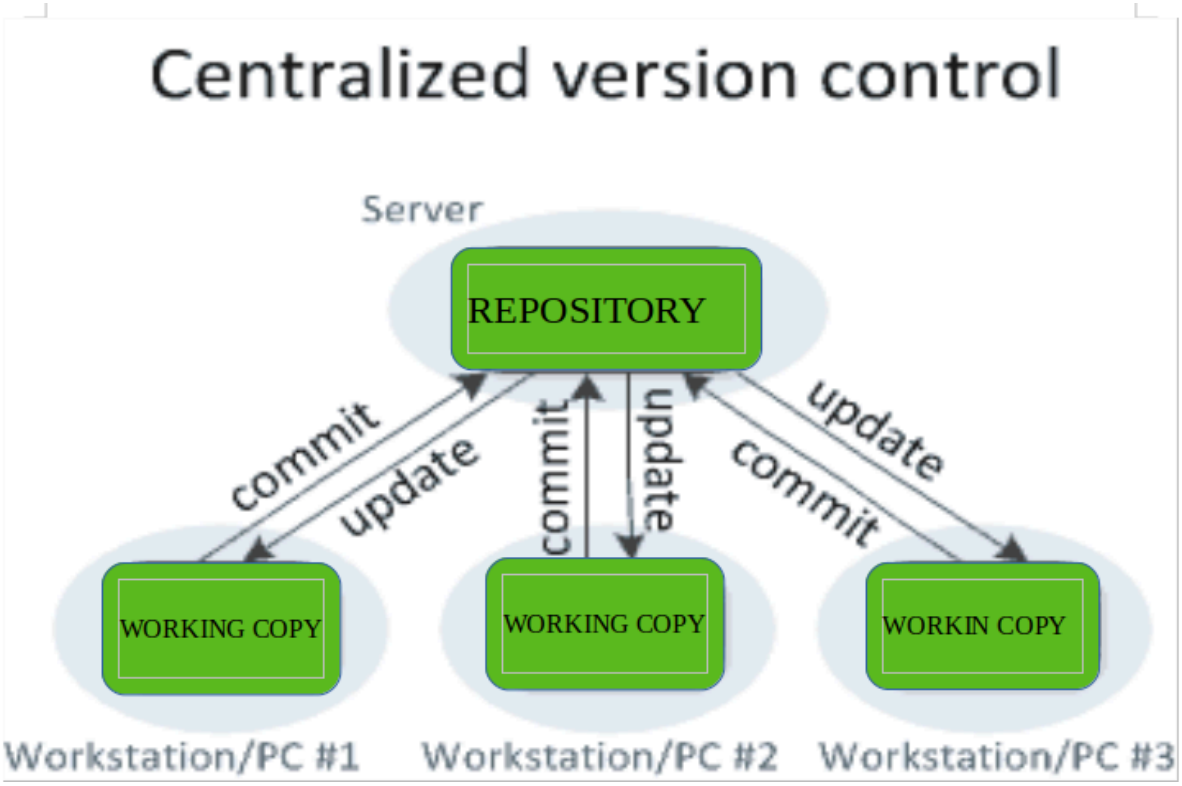
- Local Version Control Systems
- Centralized Version Control Systems
- Distributed Version Control Systems

Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

Centralized Version Control Systems: Centralized version control systems contain just one repository globally and every user need to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.

Two things are required to make your changes visible to others which are:

- You commit
- They update



The benefit of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.

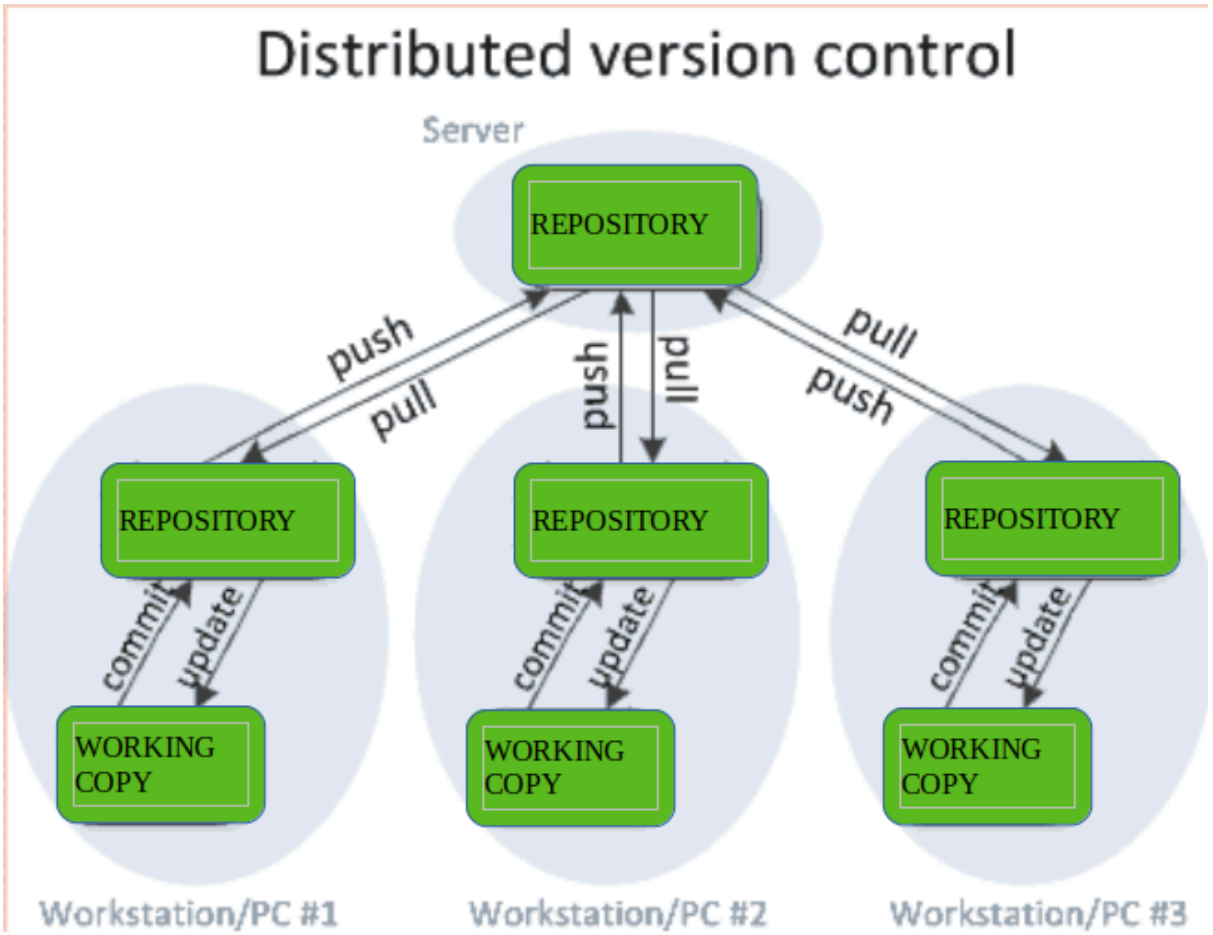
It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven't been kept? You lose absolutely everything.

Distributed Version Control Systems: Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get others' changes unless you have first pulled those changes into your repository.

To make your changes visible to others, 4 things are required:

- You commit
- You push
- They pull
- They update

The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.



Purpose of Version Control:

- Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.
- It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
- It integrates the work that is done simultaneously by different members of the team. In some rare cases, when conflicting edits are made by two people to the same line of a file, then human assistance is requested by the version control system in deciding what should be done.
- Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

Practical:

git init:

- Initializes a new Git repository. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.

git init

git clone:

- Creates a copy of a remote repository on your local machine.

git clone <repository_url>

git add:

- Adds changes in the working directory to the staging area, preparing them for the next commit.

git add <file>

git commit:

- Records changes to the repository. It captures a snapshot of the project's currently staged changes.

git commit -m "Your commit message"

git status:

- Shows the status of changes as untracked, modified, or staged.

git status

git pull:

- Fetches changes from a remote repository and merges them into the current branch.

git pull origin <branch>

git push:

- Pushes committed changes to a remote repository.

git push origin <branch>

git branch:

- Lists all local branches in the current repository.

git branch

git checkout:

- Switches between branches or restores working tree files.

git checkout <branch>

git merge:

- Combines changes from different branches.

git merge <branch>

git log:

- Displays a history of commits.

git log

git remote:

- Shows a list of remote repositories associated with the current repository.

git remote -v

git fetch:

- Downloads changes from a remote repository without merging.

git fetch origin

git diff:

- Shows changes between commits, commit and working tree, etc.

git diff

git reset:

- Resets the staging area to the most recent commit.

git reset

git rm:

- Removes files from the working directory and stages the removal.

git rm <file>

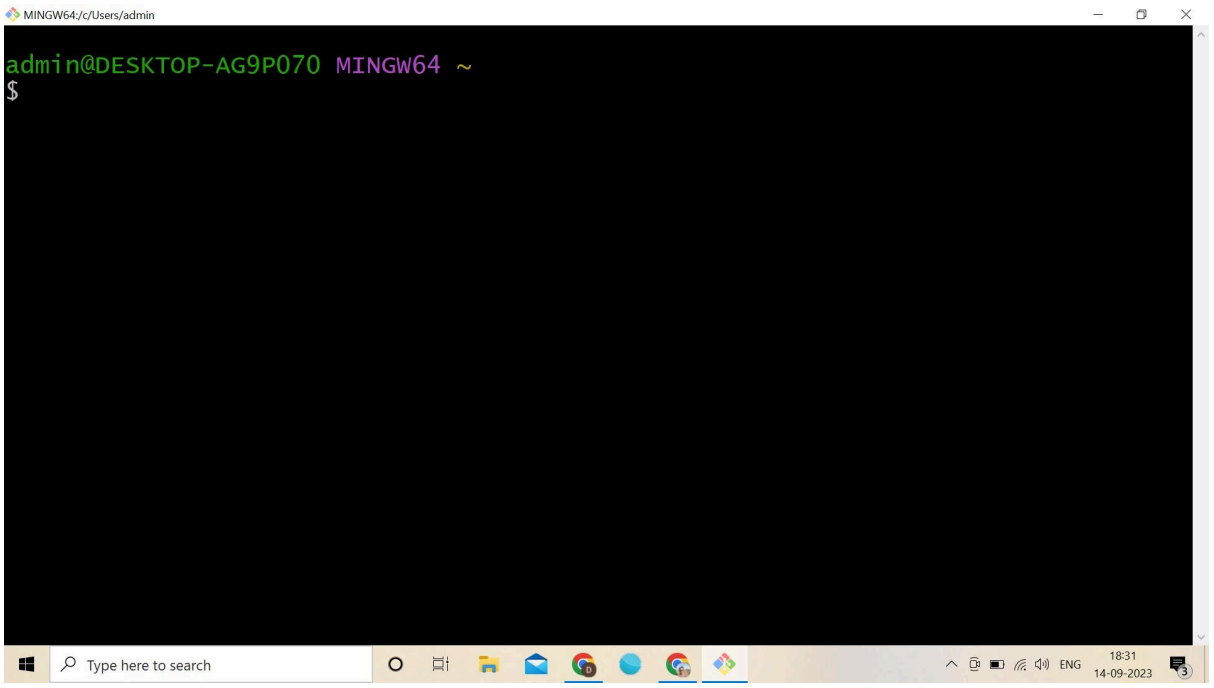
git tag:

- Marks a specific commit with a tag to make it easier to reference.

git tag -a v1.0 -m "Release version 1.0"

These are just a few of the many Git commands available. You can use git --help or git <command> --help for more detailed information on any specific command. Additionally, refer to the official Git documentation for comprehensive guidance.

Once installed git,**To launch Git Bash open the Windows Start menu, type *git bash* and press Enter (or click the application icon).**



Step 2:

Check the Git version: **$ git --version**

Step 3:
For any help, use the following command: **$ git help config**

This command will lead you to a browser of config commands. Basically, the help command provides a manual from the help page for the command just following it (here, it's config).

Another way to use the same command is as follows: **$ git config --help**
Step 4:
Create a local directory using the following command: **$ mkdir AIDS**
$ cd AIDS
Step 5:Put ur ML Model in given folder
The next step is to initialise the directory: **$ git init**
Step 6:

Go to the folder where "AIDS" is created and upload exp.no.1's login page "Loginpage.html"

Step 7:

Enter the Git bash interface and type in the following command to check the status: **$ git status**

Step 8:

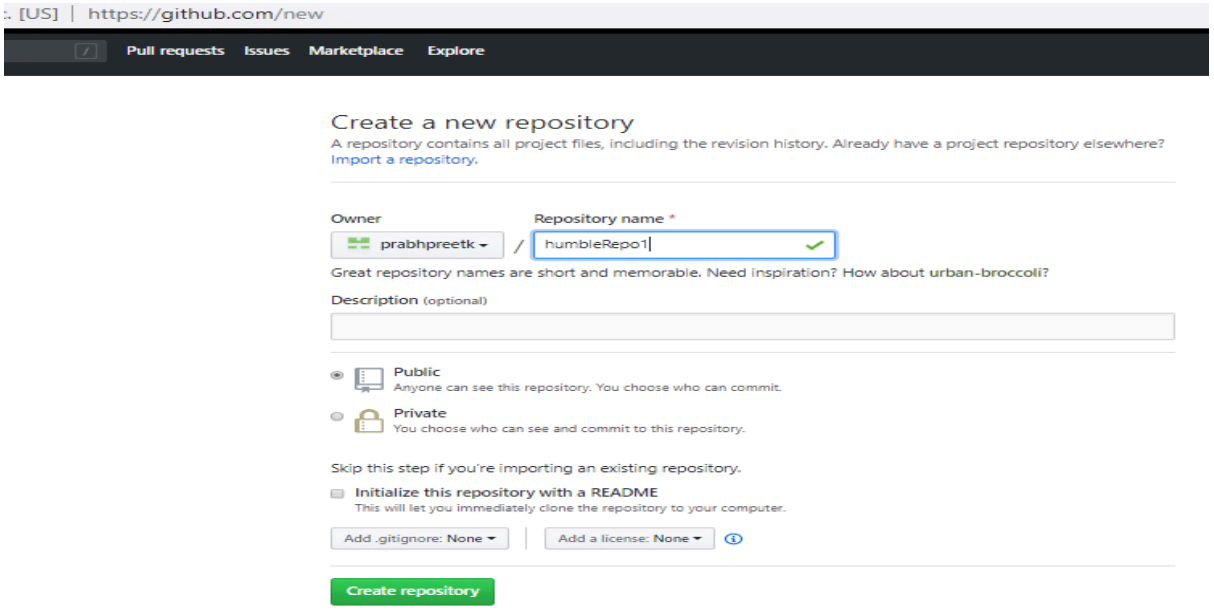
Add the "loginpage" to the current directory using the following command: **$ git add .**

Step 9:(first time user tobe followed step.10)

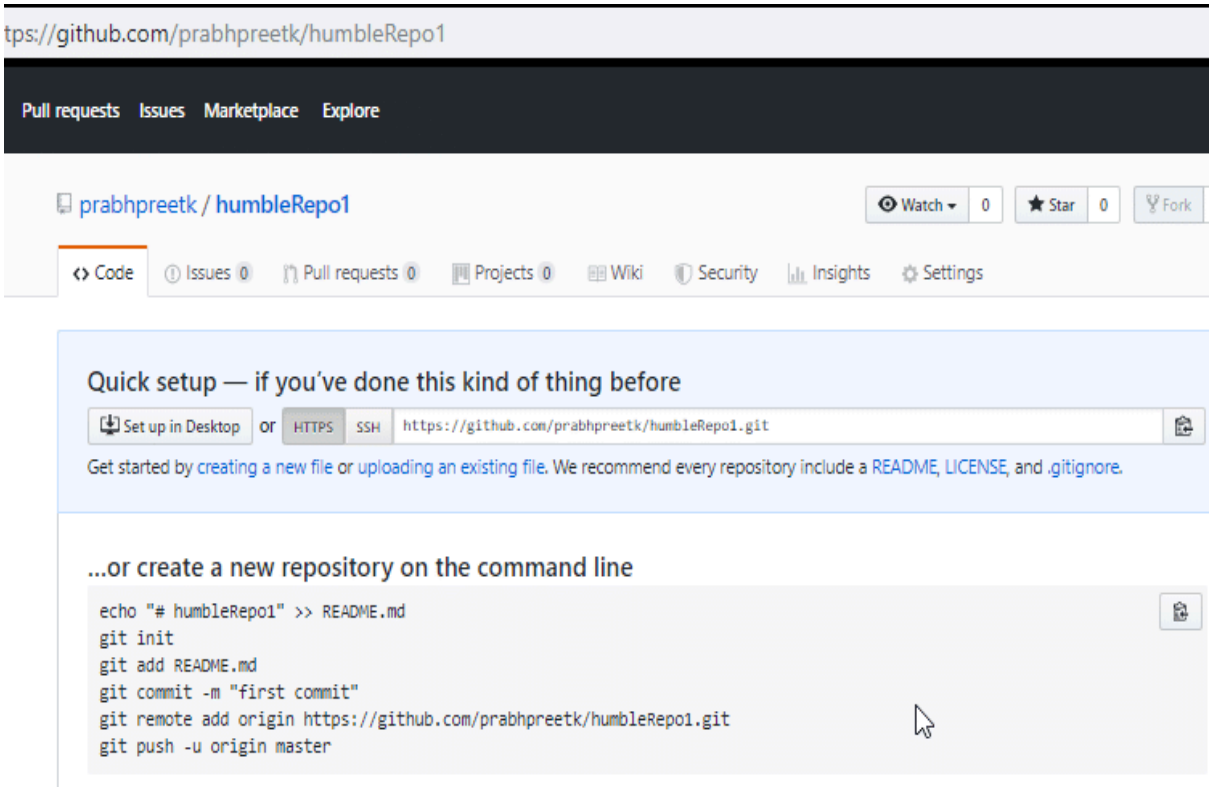
Next, make a commit using the following command:
$ git commit -m "committing a text file"
Comment

Step 10:

Open your Github account and create a new repository with the name **"AIDS_demo" or any name** and click on "Create repository." This is the remote repository.



Next, copy the link of "URL"



Step 11:

Go back to Git bash and link the remote and local repository using the following command:
$ git remote add origin <URL>
Step 12: Push the local file onto the remote repository using the following command:
$ git push origin master

Step 13: Move back to Github and click on **"AIDS_demo"** and check if the local file **"MlModel"** is pushed to this repository

3.b. Learn to track code changes, collaborate with team members, and manage different branches.

Check Repository Status:

Check the status of your repository:
git status

Create and Switch Branches:

Create a new branch:
git branch <branch-name>
Switch to a branch:
git checkout <branch-name>
Or, in recent Git versions:
git switch <branch-name>
Merge Branches:

Merge changes from one branch to another:
git merge <branch-name>

View Commit History:

View commit history:
git log

Collaborate with Remote Repositories:

Add a remote repository:

git remote add origin <remote-url>

Push changes to a remote repository:

git push -u origin <branch-name>

Pull changes from a remote repository:

git pull origin <branch-name>

Handle Merge Conflicts:

If conflicts occur during a merge, resolve conflicts manually, then:

git add <conflicted-file>
git merge --continue

These commands cover the basics of working with Git for version control. Remember to replace <filename>, <branch-name>, and <remote-url> with the actual file name, branch name, and remote repository URL, respectively.

References:
https://www.geeksforgeeks.org/version-control-systems/
https://www.perforce.com/blog/vcs/what-is-version-control