**EXPERIMENT-1**

**AIM: Version control with Git**

What is Version and what are its types?

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.

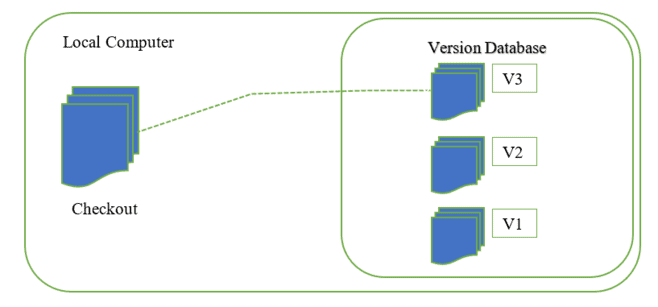
A version control system is a tool that helps you manage “versions” of your code or changes to your code while working with a team over remote distances . Version control keeps the track of every modification in special kind of database that is accessible to version control software.

It helps you to revert back to older version just in case a bug or issue is introduced to the system or fixing a mistake without disrupting the work of other team members.

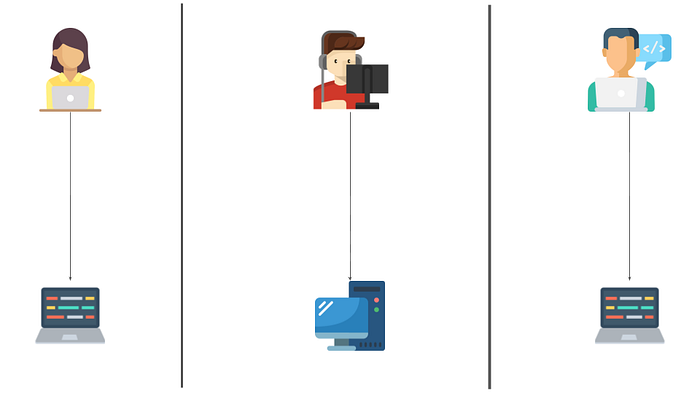
You can compare changes and see who last modified something that might be causing a problem, who introduced an issue and when, and more.

**Type of VCS**

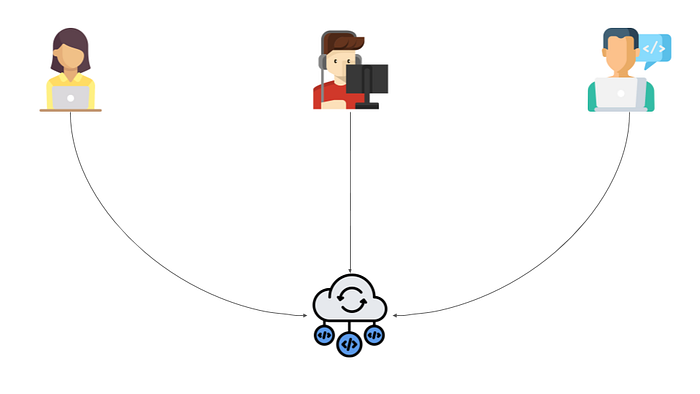
1. Local version control system
2. Centralized version control system
3. Distributed version control system



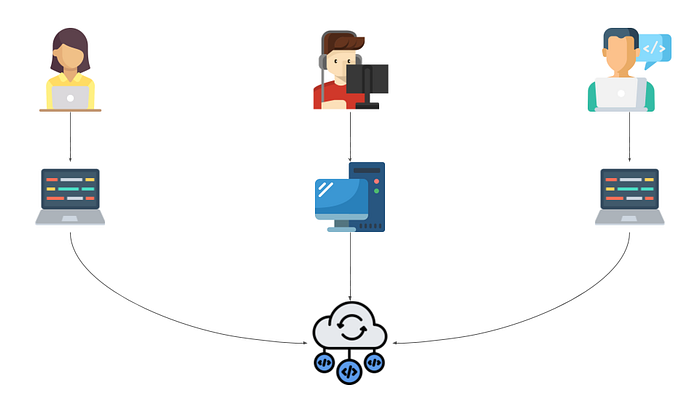
**1. Local Version Control Systems:** You manage and version all the files only within your local system. There is no remote server in this scenario. All the changes are recorded in a local database. If local machine crashes , it would not be possible to retrieve the files and all the information will be lost. As we can see in the image below, every developer has their own computers and are not sharing anything.



**2. Centralized Version Control Systems:** Here, there’s a central repo shared with all the developers, and everyone gets their own working copy. Whenever you commit, the changes get reflected directly in the repo. Unlike distributed systems, developers directly commit to the remote. Which means they may affect files knowingly or unknowingly.



**3. Distributed Version Control Systems:** In distributed systems, there is a local copy of the repo for every developer on their computers. They can make whatever changes they want and commit without affecting the remote repo. They first commit in their local repo and then push the changes to the remote repo. This is the type used majorly today.



**GIT (VCS)**

Git is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development.

Git is DevOps used for source code management. It is a free and open-source version control system used to handle small to very large projects efficiently. Git is used to tracking changes in the source code, enabling multiple developers to work together on non-linear development. Linus Torvalds created Git in 2005 for the development of the [Linux](https://www.simplilearn.com/linux-programming-for-beginners-article) kernel.



Git is a speedy and efficient distributed VCS tool that can handle project of any size from small to very large ones. Git provide cheap local branching , convenient staging area and multiple workflows. It is free, open source software that lower the cost because developer can use git without paying money. It provide support for non-linear development . Git enable multiple developer or teams to work

Separately without having impact on the work of other .

**GIT HUB**

Git hub is a for profit company that offers a cloud-based git repository hosting service. Essentially, it makes it a lot easier for individuals and teams to use Git for version control and collaboration.

Git hub’s interface is a user friendly enough so even novice coders can take advantage of git. Without Git hub, using Git generally requires a bit more technical savvy and use of the command line.

It is the world’s biggest open source software developer community platform where the user upload their project using software Git.

**PROBLEM STATEMENT**

In today’s time, it creates a lot of hustle to select a suitable car while buying it . It sometimes is really a time consuming process to find out

* what type of car you want
* what should be the body type of car
* what should be the transmission of the car and
* what should be the fuel type of the car

under a given Budget.

**SOLUTION**

The given problem has been solved using a python based project :

**CAR FILTER**

Its main aim is to recommend suitable cars to the user as per the information entered by them based on budget range of the user, fuel type of the car they want, transmission type and what body type is desired by the user along with addition and deletion of data of such cars from original data.

* Car filter project recommends the most suitable cars to the user as per their requirements.
* It makes it easy for the user to select over a wide range of cars.
* It provides necessary information of suggested cars along with the budget.
* It can add data of cars to the existing data .
* It can remove data of particular cars from the existing data.

**OBJECTIVE**

The purpose of the project ‘Car Filter’ is to recommend the user with suitable cars based on the requirements entered by the user.

* This project is based on the idea to create ease at selecting a car out of a wide range .
* The user can also add such data (budget, body type, fuel type and transmission type )to the database.
* It creates a lot of hustle while selecting a suitable car and aims at suggesting suitable cars as per the requirements of the user including budget range ,in which the user has to buy the car ,body type, fuel type and transmission of the cars available in the given range .
* This approach can also be used to add data to the database of this project .

**Resource Requirements- Front End**

**TOOL USED:**

Jupyter NOTEBOOK, VS CODE

RESOURCE REQUIREMENT:

In this project we dealt with creating filters in python , which are used to recommned cars to the user under certain conditions.

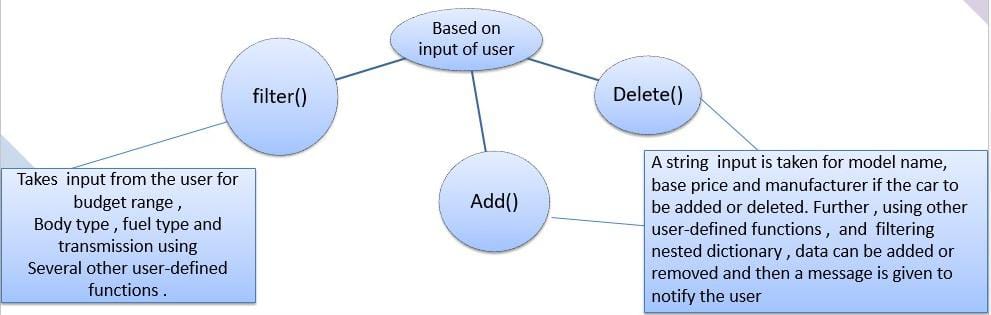
**Insights to the program:**

HIGH LEVEL APPROACH:

* In this project, the system first asks the user whether they have to search for a car or add data or delete data or exit the program.
* The user needs to enter the option number as per the requirement.
* for option [1] i.e. “SEARCH FOR A CAR”……..system further asks the budget range, body type, Transmission and fuel type of the car and suggests cars satisfying the given conditions .
* for option [2] i.e. “ADD DATA “…………system asks for information of car to be added.
* for option [3] i.e. “DELETE DATA” ……system asks for information of car to be deleted.
* for option [4] i.e “EXIT”……….the program will end.

LOW LEVEL APPROACH:

* Creating a nested dictionary , containing all the data of budget, body type , transmission, fuel type of the cars .
* A menu is printed in an infinite loop asking whether the user wants to search for a car, add data, delete data or exit.
* According to the choice entered by the user, program splits into 3 functions as follows:



CONCEPTS AND COMMANDS

In the current era, the open-source contribution is at its peak. It’s getting so much popular among students and there are also some open-source programs and competitions to learn so much about development. Open source is a great way to learn and make a better community. There are so many organizations where one can contribute and learn. For that one have to know about using git commands

* **git add:** Adds files to the staging area for Git. Before a file is available to commit to a repository, the file needs to be added to the Git index (staging area).
* **git add --all:** To add all files of the current directory to the staging area
* **git add -A:** To add all text files of the current directory to the staging area.
* **git commit -m “Initial commit”:** To add any of our tracked files to the staging area and commit them by providing a message to remember.
* **git diff:** Diff command is used in git to track the difference between the changes made on a file.
* **git diff –staged**: diff of what is staged but not yet committed
* **git status:** used to display the state of the repository and staging area. It allows us to see the tracked, untracked files and changes. This command will not show any commit records or information.
* **git log:** To show the chronological commit history for a repository. This helps give context and history to a repository.
* **git log –oneline**: Shows all the log and history of our commits in one line.
* **git log –oneline – graph**: Shows a network graph of all our commits
* **ls:** To see directories and files in the current directory
* **git branch:** To see all the branches present and the current branch that we are working on
* **git checkout<branchname>:** To switch to a branch example ‘act1’from master branch
* **git checkout -b <branchname>:** This command creates a new branch and also switches to it
* **git merge<branchname> :** To merge a branch example ‘act1’ with the master branch.
* **git branch -D <branchname>:** To forcefully delete a branch without making commits.
* **git branch –d <branchname>:** This is also used to delete a branch. The -d option will delete the branch only if it has already been pushed and merged with the remote branch.
* **git remote add <address>:** To add new remotes to our local repository for a particular git address
* **git push:** Sends local commits to the remote repository. *git push* requires two parameters: the remote repository and the branch that the push is for.
* **git push – u origin master:** To push all the contents of our local repository that belong to the master branch to the server(Global repository).
* **git pull:** To get the latest version of a repository run *git pull*. This pulls the changes from the remote repository to the local computer.
* **git config:** To set the basic configurations on Github like your name and email.
* **git config –user.name “XYZ”:** Sets configuration values for your username on git
* **git config –user.email xyz@gmail.com:**Sets configuration values for your user email on git.
* **git init:** This command turns a directory into an empty Git repository.
* **git clone<url>:** To clone or make a local copy of the global repository in your system
* **git merge**: Integrate branches together .git merge combines the changes from one branch to another branch. For example, merge the changes made in a staging branch into the stable branch
* **git remote:** To connect a local repository with a remote repository. A remote repository can have a name set to avoid having to remember the URL of the repository.
* **git remote -v:** Shows URLs of remote repositories when listing your current remote connections.
* **git reset:** git reset is used when we want to unstage a file and bring our changes back to the working directory. git reset can also be used to remove commits from the local repository.

**git reset has 3 variants:**

§ ***git reset –soft HEAD~1*** : This command will remove the commit but would not unstage a file. Our changes still would be in the staging area.

§ ***git reset –mixed HEAD~1*** or ***git reset HEAD~1:*** This is the default command that we have used in the above example which removes the commit as well as unstaged the file and our changes are stored in the working directory.

§ ***git reset –hard HEAD~1 –*** This command removes the commit as well as the changes from your working directory.

* **git revert**: This command is used to remove the commits from the remote repository. Since now our changes are in the working directory, let’s add those changes to the staging area and commit them.
* **git ignore :**Creating a file with the name .gitignore.The files in that directory and deeper folders that match the patterns in the file will be ignored by gitignore.
* **git help:** Take help from the Github section for different commands and other errors
* **mkdir dir :** Create a directory with the name ‘dir’ if not created initially
* **cd dir :** To go inside the directory ‘dir’ and work its contents.
* **git commit -m “Initial commit”:** To commit our changes(taking a snapshot) and providing a message to remember for future reference.
* **git remote rm:** To remove a remote from our local repository.
* \**git rm <filename>:** This command deletes the file from your working directory and stages the deletion.

**CREATE A DISTRIBUTED AND ADD MEMBERS IN PROJECT TEAM**

* Login to your git hub account and you will land on the homepage as shown below. Click on repositories option in the menu bar.
* Click on “New” button in the top right corner.
* Enter the Repository name and add the description of the repository.
* Then select the public or private repository option, it is recommended to select the public option to collaborate with teammates.
* If you want to import code from an existing repository select the code option.
* Now click on “create repository”
* Now, you have created your repository successfully.
* To add members to your repository open your repository and select the settings option in the navigation bar.
* After clicking on collaborators Github asks you to enter your password to confirm the access to the repository.After entering the password you can manage access and add/remove team members to your project.
* To add members click on the add people option and search for the id of your respective team member.
* To accept the invitation from your team member, open your email registered with Github.
* You will receive an invitation mail from the repository owner. Open the email and click on accept invitation.
* You will be redirected to Github where you can either select to accept or decline the invitation.
* You will be shown the option that you are now allowed to push.
* Now all members are ready to contribute to the project.

**OPEN AND CLOSE A PULL REQUEST**

* To open a pull request we first have to make a new branch by using git branch branchname option.
* After making new branch we add a file to the branch or make changes in the existing file.
* Add and commit the changes to the local repository.
* Use the git push origin branchname option to push the new branch to the main repository.
* After pushing new branch github will either automatically ask you to create a pull request or you can create your own pull request.
* To create your own pull request, click on pull request option.
* Github will detect any conflict and ask you to enter a description of your pull request.
* After opening a pull request all the team members will be sent the request if they want to merge or close the request.
* If the team member chooses not to merge your pull request then they will close the pull request.
* To close the pull request simply click on close pull request and add comment/reason why you closed the pull request.
* You can see all the pull request generated and how they were dealt with by clicking on pull request option.

**Create a pull request on a team member’s repo and close pull requests generated by team members on their Repo as a maintainer**

To create a pull request on a team member’s repository and close requests by any other team members as a maintainer follow the procedure given below:-

* \Do the required changes in the repository, add and commit these changes in the local repository in a new branch.
* Push the modified branch using git push origin *branch-name*.
* Open a pull request by following the procedure from the above experiment.
* The pull request will be created and will be visible to all the team members.
* Ask your team member to log into his/her Github account.
* They will notice a new notification in the pull request menu.
* Click on it. The pull request generated by you will be visible to them**.**
* By selecting the merge branch option the main branch will get updated for all the team members.
* By selecting close the pull request the pull request is not accepted and not merged with the main branch.
* The process is similar to closing and merging the pull request you. It simply includes an external party to execute.
* The result of merging the pull request is shown below.

**Adding the Gitignore to the repository**

* Create a file from git bash using touch .gitignore
* It will create a file named .gitignore , it act as a text editor, whichever file name or file type is added to it will be ignored and not tracked.
* Then add this to your repo by pushing it to your git account.

**PUBLISH AND PRINT NETWORK GRAPHS**

The network graph is one of the useful features for developers on GitHub. It is used to display the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network.

A repository's graphs give you information on traffic, projects that depend on the repository, contributors and commits to the repository, and a repository's forks and network. If you maintain a repository, you can use this data to get a better understanding of who's using your repository and why they're using it.

Some repository graphs are available only in public repositories with GitHub Free:

* Pulse
* Contributors
* Traffic
* Commits
* Code frequency
* Network

**Steps to access network graphs of respective repositories**

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click  **Insights**.
3. 3.At the left sidebar, click on **Network**

You will get the network graph of your repository which displays the branch history of the entire repository network, including branches of the root repository and branches of forks that contain commits unique to the network.

**The network graph from collaborator.**

## **Listing the forks of a repository**

Forks are listed alphabetically by the username of the person who forked the repository

Clicking the number of forks shows you the full network. From there you can click "members" to see who forked the repo

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click  **Insights**.

In the left sidebar, click **Forks**.

Here you can see all the forks

**We** can also see the graph from our git bash terminal using the command **git log -–graph**

**REFERENCES**