Name: Sarthak Hinge
Roll No: 35
Batch: T-12

EXPERIMENT NO: 01

AIM: To understand Version Control System, install Git and create a GitHub Account.

THEORY:

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. As development environments have accelerated, version control systems help software teams work faster and smarter. They are especially useful for DevOps teams since they help them to reduce development time and increase successful deployments.

1. A complete long-term change history of every file. This means every change made by many individuals over the years. Changes include the creation and deletion of files as well as edits to their contents. Different VCS tools differ on how well they handle renaming and moving of files. This history should also include the author, date and written notes on the purpose of each change. Having the complete history enables going back to previous versions to help in root cause analysis for bugs and it is crucial when needing to fix problems in older versions of software. If the software is being actively worked on, almost everything can be considered an "older version" of the software.

2. Branching and merging. Having team members work concurrently is a no-brainer, but even individuals working on their own can benefit from the ability to work on independent streams of changes. Creating a "branch" in VCS tools keeps multiple streams of work independent from each other while also providing the facility to merge that work back together, enabling developers to

verify that the changes on each branch do not conflict. Many software teams adopt a practice of branching for each feature or perhaps branching for each release, or both. There are many different workflows that teams can choose from when they decide how to make use of branching and merging facilities in VCS.

3. Traceability. Being able to trace each change made to the software and connect it to project management and bug tracking software such as Jira, and being able to annotate each change with a message describing the purpose and intent of the change can help not only with root cause analysis and other forensics. Having the annotated history of the code at your fingertips when you are reading the code, trying to understand what it is doing and why it is so designed can enable developers to make correct and harmonious changes that are in accord with the intended long-term design of the system. This can be especially important for working effectively with legacy code and is crucial in enabling developers to estimate future work with any accuracy.

Types of Version Control Systems:

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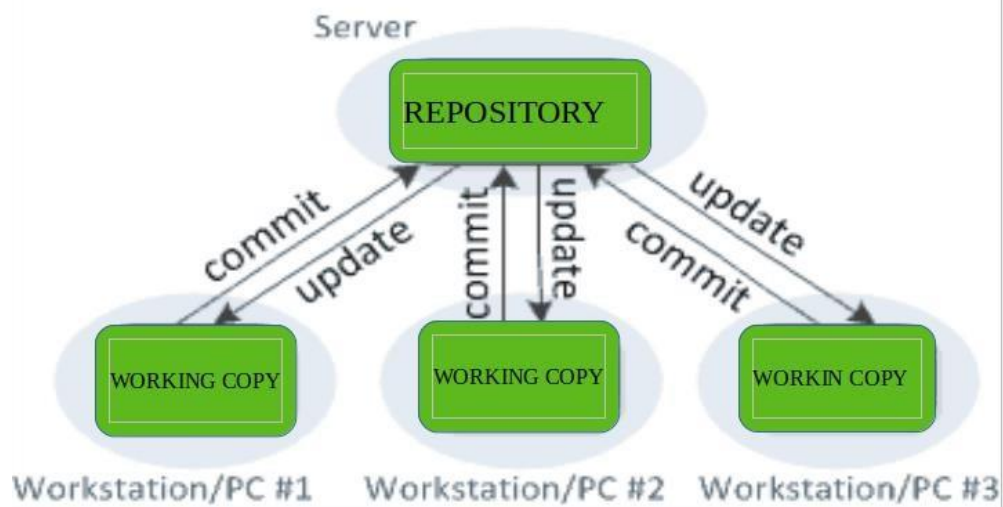- Distributed Version Control Systems

1. **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.

2. **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

Centralized version control

3. **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.

Distributed version control
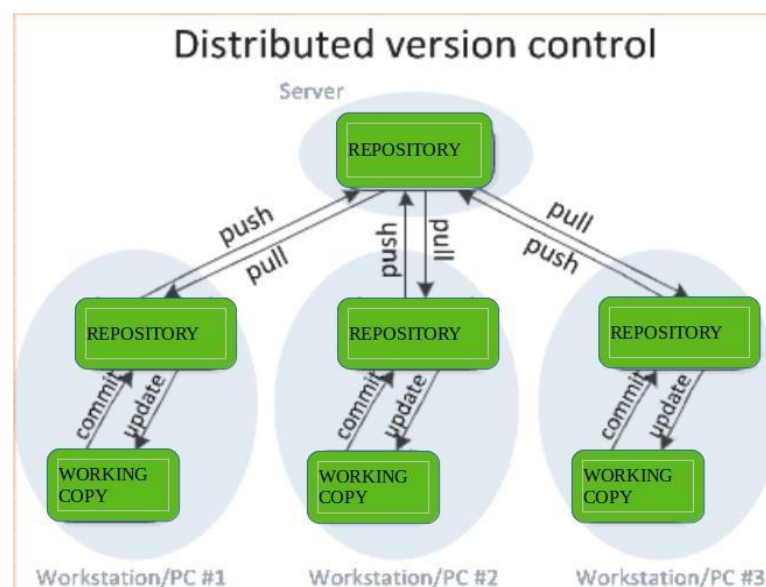
Git:

Git is a distributed version control system (VCS) that's widely used for tracking changes in source code during software development. It's designed to handle everything from small to very large projects with speed and efficiency. Here are some key points about Git:

1. **Version Control**: Git allows multiple developers to work on the same project at the same time, keeping track of each change made to the code. This makes it easy to go back to previous versions of the code or to collaborate with others without overwriting each other's work.

2. **Distributed Nature**: Unlike centralized version control systems (like Subversion), Git is distributed. This means every developer has a full copy of the repository (the complete project history) on their local machine, not just a snapshot of the current state. This makes operations like committing, branching, and viewing history faster and more reliable.

3. **Branches**: One of Git's most powerful features is branching. You can create a branch to work on a feature or fix a bug, and then merge it back into the main codebase when you're done. This allows developers to work independently without interfering with each other's progress.

4. **Commit and History**: A commit in Git is a snapshot of the project at a particular point in time. Every commit is unique and is identified by a hash (a long string of numbers and letters). You can use Git commands to explore the history of changes, view who made what changes, and when.

5. **Remote Repositories**: Git can work with remote repositories, like GitHub or GitLab, allowing teams to collaborate and store code online. You can push your local commits to the remote repository, and pull down others' changes to keep your local copy up to date.

6. **Merging and Conflict Resolution**: When two people work on the same part of the code, a conflict might occur when trying to merge their changes. Git provides tools to resolve these conflicts, either manually or through a merge tool.

7. **Popular Commands**: Some common Git commands include:

   o git clone: Make a copy of a repository.

   o git status: See which files have been modified or are untracked.

- git add: Add files to the staging area (preparing them to be committed).

- git commit: Commit changes to the local repository.

- git push: Push changes to a remote repository.

- git pull: Fetch changes from a remote repository and merge them into your local branch.

- git merge: Merge branches together.

8. **Rebasing:**

- **Rebase** is an alternative to merging. Instead of merging changes, rebasing allows you to move or reapply commits from one branch onto another.

- Rebasing is useful for creating a cleaner history by avoiding merge commits, but it can rewrite commit history, which should be used cautiously—especially with shared branches.

- Example: git rebase main will take the changes in your current branch and "reapply" them on top of the latest main branch.

Git's flexibility and power make it a fundamental tool for developers working on both small and large projects. If you're new to it, there are a lot of great resources and tutorials to get you started.

GitHub:

GitHub is a platform that hosts Git repositories and provides a suite of tools for collaborative software development. It builds on top of Git, offering additional features for project management, team collaboration, and code sharing. Here are some key features and aspects of GitHub:

1. Remote Repository Hosting:

GitHub allows you to store your Git repositories in the cloud. This means you can have a backup of your code, share it with others, and collaborate on projects, even if your local machine crashes or you're working on different devices.

2. Collaboration:

GitHub is widely used for team collaboration. It allows multiple developers to work on the same codebase without overwriting each other's work. Developers can fork a repository (make their own copy), work on it independently, and then submit a pull request (PR) to propose changes to the original project. Other collaborators can review, comment on, and merge those changes.

3. Pull Requests (PR):

Pull requests are a key feature of GitHub, enabling users to propose changes to a project. A pull request shows what changes you've made (such as new code or bug fixes) and allows other team members to review those changes before they are merged into the main codebase. It's a crucial tool for peer review and collaboration in open-source projects.

4. Issues and Project Management:

GitHub includes issue tracking to manage bugs, feature requests, and tasks. Issues can be tagged with labels (e.g., "bug", "enhancement", "help wanted"), assigned to specific people, and connected to specific commits or pull requests. You can organize issues into "projects," which is a tool for planning, tracking progress, and keeping things on track.

5. Forking and Contributing:

GitHub encourages open-source collaboration. If you find a project you'd like to contribute to, you can fork it (copy it to your GitHub account), make changes, and then create a pull request. This allows open-source contributors to share code and improvements without needing direct access to the original project.

6. GitHub Pages:

GitHub provides a free hosting service called **GitHub Pages** for static websites. You can host personal websites, blogs, or project documentation directly from a GitHub repository.

7. GitHub Actions:

GitHub Actions is a CI/CD (Continuous Integration and Continuous Deployment) service that allows you to automate tasks in your development workflow, such as running tests, building applications, or deploying code. You can create workflows that are triggered by specific events (e.g., when code is pushed or a pull request is created).

8. Social Features:

GitHub functions as a social network for developers. You can follow other developers, star repositories (to bookmark or show appreciation), and contribute to discussions. This makes it not only a version control system but also a community hub for code sharing and networking.

9. GitHub Marketplace:

GitHub has a marketplace where you can find third-party tools and integrations to enhance your workflow, such as automated testing tools, code quality checkers, and project management add-ons.

10. Security and Access Control:

GitHub allows you to set up private repositories (where only invited collaborators can see or contribute to the code) or public repositories (open to everyone). It also offers features like protected branches, where certain branches (e.g., the main branch) are restricted from being changed unless certain checks are passed, providing additional security and stability to the project.

11. GitHub Copilot:

GitHub Copilot is an AI-powered code completion tool that can help developers write code more efficiently. It provides code suggestions and snippets directly inside your editor (e.g., Visual Studio Code).

12. Community and Open Source:

GitHub is the home of millions of open-source projects. Many developers and companies host their projects there, making it the largest collection of open-source code in the world. It's a great place to learn from other developers, contribute to existing projects, and share your own code with a global community.

In short, **GitHub** is like a social coding platform that extends Git's version control with collaboration features, tools for managing projects, and a global community for open-source and team-based development. It's a powerful tool for developers, both individual and corporate, to build, share, and maintain software.

Steps to create a GitHub account:

Step 1: Fill the required details.

Step 2: Enter the verification code from the email.

Step 3: Your GitHub account is ready.
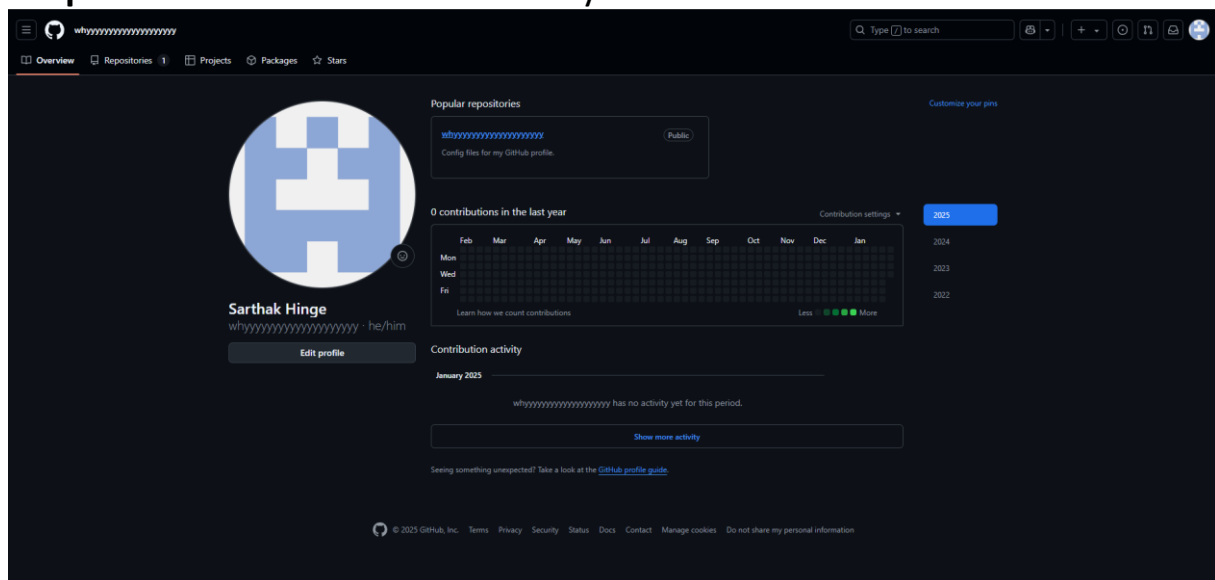
CONCLUSION:

Thus, we have successfully implemented Version Control System, the process of installing Git and creating a GitHub Account.