Version Control System

- A Version Control System (VCS) is a software tool that helps in managing changes to source code over time.
- It allows multiple developers to work on a project simultaneously.
- In simple words, version control system is a tool that helps to track changes in project code.

Types of VCS

Centralized Version Control System (CVCS)

- Uses a central server to store all files and enables collaboration.
- Example: CVS (Concurrent Versions System), Subversion (SVN).

Distributed Version Control System (DVCS)

- Each user has a complete copy of the repository, including its history.
- Example: Git, Mercurial.

git

- Git is a distributed version control system (VCS) designed to handle everything from small to very large projects with speed and efficiency.
- It was created by Linus Torvalds in 2005 and has since become the most widely used version control system in the world.
- Git is open source and is commonly used for source code management in software development, but it can be used to keep track of changes in any set of files.
- Git has become an essential tool in modern software development, providing a robust and flexible platform for version control and collaboration.

key concepts and features of Git

Version Control System (VCS): Git allows multiple developers to collaborate on a project simultaneously. It keeps track of changes to source code or any other set of files over time.

Distributed System: Unlike centralized version control systems, every Git working directory is a full-fledged repository with complete history and version tracking capabilities. This means that each developer has a copy of the entire project with its history on their local machine.

Commits: In Git, a commit represents a set of changes to the code. Each commit has a unique identifier and a commit message that describes the changes made.

Branching: Git encourages the use of branches, which are separate lines of development. Developers can create branches to work on specific features or fixes without affecting the main codebase. Branches can be easily merged when the work is complete.

Merging: Git makes it easy to merge changes from one branch into another. This is particularly useful when multiple developers are working on different features concurrently.

Remote Repositories: Git facilitates collaboration by allowing developers to clone repositories from a remote source [eg. github]. This is commonly done using services like GitHub, GitLab, or Bitbucket.

Pull Requests: On platforms like GitHub, developers can propose changes to a repository by submitting a pull request. This allows others to review the changes before they are merged into the main codebase.

Staging Area (Index): Git has a staging area where changes are prepared before being committed. This allows developers to selectively commit changes rather than committing everything at once.

Git Commands: Some common Git commands include git init (initialize a new repository), git clone (clone a repository), git add (add changes to the staging area), git commit (commit changes), git pull (fetch changes from a remote repository), and git push (push changes to a remote repository).

GitHub

- GitHub is a web-based platform that uses Git for version control and is widely used for software development.
- It provides a hosting service for software development projects using the Git version control system.
- GitHub offers a collaborative environment where developers can work on projects, contribute to open-source software, and manage code repositories.
- GitHub has become a central platform for software development, particularly for open-source projects.
- Many developers and organizations use GitHub to manage their code repositories, collaborate with others, and contribute to a wide range of projects.
- In simple words, GitHub is a website that allows developers to store and manage their code using git.

https://github.com

Key aspects of GitHub

Code Hosting: GitHub allows developers to host their Git repositories on the platform. This means that you can store your project's source code, track changes, and collaborate with others on GitHub.
Collaboration: GitHub is designed to facilitate collaboration among developers. Multiple people can work on the same project simultaneously by creating branches, making changes, and proposing those changes through pull requests.
Pull Requests: A pull request (PR) is a feature that enables developers to propose changes to a repository. It allows others to review the changes, discuss them, and, if approved, merge them into the main codebase.
Issue Tracking: GitHub provides a built-in issue tracking system. Developers can use issues to report bugs, request new features, or discuss any aspect of the project. Issues can be linked to pull requests, providing a comprehensive view of project development.
Wiki and Documentation: GitHub allows you to create wikis and documentation for your projects. This is useful for providing information about the project, setting up guidelines, and helping contributors understand how to contribute.
GitHub Actions: GitHub Actions is a feature that automates workflows. Developers can define custom workflows, such as running tests or deploying applications, and GitHub will automatically execute these workflows in response to events like pushes or pull requests.

GitHub Pages: GitHub Pages is a feature that allows you to host static websites directly from your GitHub repository. It's commonly used for project documentation or personal/professional websites.

Community and Social Coding: GitHub is a hub for open-source development, and it encourages social coding. Developers can follow projects, star repositories, and contribute to projects created by others. It fosters a sense of community and collaboration.

Integration with Other Tools: GitHub integrates with various development tools, including continuous integration services, code quality analyzers, and project management tools.

Security Features: GitHub provides security features such as code scanning, dependency tracking, and alerts for known vulnerabilities to help developers build more secure software.

Importance of Git & GitHub

- Git and GitHub play crucial roles in modern software development, offering tools and platforms that enhance collaboration, code quality, and overall project management.
- Their importance extends beyond version control to encompass collaboration, community engagement, and automation in the software development lifecycle.

Importance of Git

Version Control

History Tracking: Git allows developers to track changes in the codebase over time, providing a detailed history of who made what changes and when.

Rollback and Revert: Git enables the easy rollback to previous states or the reverting of specific changes, providing a safety net for development.

Collaboration

Parallel Development: Git supports branching, enabling multiple developers to work on different features or fixes simultaneously without interfering with each other.

Merging: Developers can merge their changes back into the main codebase seamlessly, ensuring a smooth collaboration process.

Flexibility

Staging Area: The staging area allows developers to selectively commit changes, helping in creating clean and organized commits.

Offline Work: Developers can work offline and commit changes locally before pushing them to a central repository.

Performance

Speed: Git is known for its speed and efficiency, making it suitable for both small and large projects.

Local Operations: Many Git operations are performed locally, reducing the need for constant communication with a central server.

Open-Source Community

Community Support: Git has a large and active open-source community, providing resources, tutorials, and extensions that enhance its functionality.

Standardization: Git has become a standard for version control in the software development industry.

Importance of GitHub

Centralized Hosting

Accessibility: GitHub provides a centralized platform for hosting Git repositories, making code accessible from anywhere with an internet connection.

Backup and Redundancy: GitHub serves as a backup of code repositories, offering redundancy and security against data loss.

Collaboration and Code Review

Pull Requests: GitHub's pull request feature facilitates code review, discussion, and collaboration before changes are merged into the main branch.

Issue Tracking: GitHub's issue tracking system helps in managing tasks, bugs, and feature requests, streamlining project communication.

Community Engagement

Networking: GitHub is a social platform for developers, allowing them to connect, follow projects, and contribute to a global community.

Discoverability: Developers can discover interesting projects, learn from others' code, and contribute to open-source initiatives.

Automation and CI/CD

GitHub Actions: GitHub Actions automates workflows, allowing developers to set up continuous integration, testing, and deployment processes directly from their repositories.

Integration with Third-Party Tools: GitHub integrates seamlessly with various third-party tools, enhancing the development workflow.

Documentation:

GitHub Pages: GitHub Pages allows developers to host static websites directly from their repositories, making it easy to create project documentation or showcase portfolio work.

Security and Monitoring

Security Features: GitHub provides security features such as code scanning, dependency tracking, and alerts for vulnerabilities, enhancing the overall security of projects.

Insights: GitHub provides insights into repository activity, helping project maintainers monitor contributions and user engagement.

Employability and Portfolio

Professional Profile: Many employers and recruiters look for candidates with a strong GitHub profile, showcasing their contributions to projects and involvement in the development community.

Portfolio Hosting: GitHub serves as a portfolio for developers, allowing them to showcase their projects and contributions to potential employers and collaborators.

Steps to follow

- Create a GitHub account.
- Update your profile.
- Create a new repository (public)
- Make your first commit
- Setting up git over vscode and GitHub
    - Download and install vscode
    - Download and install git according to your operating system
        - https://git-scm.com/downloads
- use command "git –version" to check the version of git
- Configuring git
    - git config --global user.name "your name"
    - git config --global user.email "your email"
    - git config --list
- cloning your repository on your local machine
    - git clone <repository link>
- view the available files in the repository using list command
    - ls
- to view the hidden files of the repository or check the .git directory
    - ls -a
- display the state / status of your code
    - git status
- apply some changes in your repository, locally then again check the status of your code.
- when you use git status, then it shows four status:
    - untracked : new files that git doesn't yet tracked
    - modified : changed status
    - staged : files are ready to be committed
    - unmodified : unchanged
- after applying changes through your local computer, you need to first staged your file and then finally commit those changes
- for staging use the command
    - git add "file_name_with_extension" - to staged a single file
    - git add . - to staged all the files
- now applying commit with a commit message
    - git commit -m "your commit message"

- now check the git status
  - git status
- after committing the changes, your repository on github would not get changed, because changes have done on your local machine.
- To apply changes the changes on your github repository you need to push your code.
  - git push origin main

Initialize git inside project directory (locally)

- We need to initialize the git repo using command

```
git init
```

- Now we have a hidden directory (.git) inside our project directory on local machine.
- Create a new project folder inside the local machine project directory using command "mkdir 'projectname' using vs-code terminal.
- Using 'cd' command enter inside the newly created project directory.

At github, create a new repository without creating any new file or readme.md file.

- The name of this newly created repository can be same or different as the local machine project directory name, but for better understanding the name should be same.
- Now add some files with some codes inside the directory.
- Now first add the files to the staging area and then commit the changes with a status message.

Now add, the origin to the remote or github.

- We need to use the command

```
git remote add origin "git_hub_new_repository_url"
```

- To verify the remote or github link, use command

```
git remote -v
```

Branches

- In Git, a branch is a lightweight movable pointer to a specific commit, which is essentially a snapshot of your project at a particular point in time.
- Each branch can represent a different line of development, and you can switch between branches to work on different features, bug fixes, or experiments.
- Branches in Git are essential for managing and organizing development efforts in a flexible and collaborative manner.
- They facilitate parallel workstreams, maintain code stability, enable version management, and enhance the overall development process.
- Their significance lies in making version control more efficient and adaptable to the needs of a project and its development team.

Branches have the following significant significance in project development:

- Isolation and Parallel Development
- Feature Development
- Bug Fixing
- Experimentation and Prototyping
- Versioning and Releases
- Collaboration
- Code Review
- Rollback and Hotfixes
- Traceability
- Experimentation and Collaboration

To check the branch status or view the branches use command

```
git branch
```

- To rename the branch, use command
  ```
  git branch – m "branch_name"
  ```
- To push the code to the remote (github) – main branch, use command
  ```
  git push origin main
  ```
  - If it is required to push the code for a long time, then you need to use the command once
    ```
    git push -u origin main
    ```
  - Then you need to just use the below shortcut command to push changes
    ```
    git push
    ```

Working with git branches

- You can create a new branch using command,

```
git checkout -b “branch-name”
```

Above command is the shorthand command for creating a new branch and switching to the new branch.

- You can also use the following commands for doing the same:

  for creating : git branch “branch-name”
  for switching : git checkout “branch-name”

since git (version 2.23), you can also use the following command to switch/navigate between branches

```
git switch “branch-name”
```

you can also use “switch” for creating a new branch and switching to the new branch.

```
git switch -c “branch-name”
```

- To delete a branch, you can use the command

```
git branch -d “branch-name”
```

Note : while deleting the branch, you always need to know that the working branch should not be the branch which you are going to delete.
The above command will delete the branch if it's fully merged into the current branch. If you want to forcefully delete a branch, use -D:

```
git branch -D “branch-name”
```

Now create two or three branches and apply changed to your files for each branch, specifically.
And see those changes while switching between branches.

Push the code / changes remotely

If you want to push your code / changes remotely for a specific branch, use command:

```
git push origin “branch-name”
```

Merge Operation

- Merging in Git is the process of combining the changes from one branch into another.
- Before merging branches, make sure you are in the branch where you want to apply changes (the target branch), and you have a clean working directory (no uncommitted changes). If you are not in the target branch, you can switch to it using git checkout target-branch.
- You can compare commits, branches, files and also find the difference in between branches by using command

```
git diff "target-branch"
```

Steps to merge changes

- **Select the Source Branch**
  You need to decide which branch you want to merge into another branch. The branch from which you merge is called the "source branch," and the branch where you want to apply changes is the "target branch."
  ```
  git checkout source-branch
  ```
- **Update the Source Branch**
  ```
  git pull origin source-branch
  ```
- **Branch merge**
  - **First Approach (Create PR or Pull Request)**
    - When you tell others about the changes which you want to push to a branch of your remote repository or GitHub repository.
    - This pull request represents a proposed set of changes to be merged into a target branch.
    - Reviewers examine the code changes, documentation, and any other materials attached to the pull request. They may comment on the code and request modifications or clarifications if necessary.
    - Once the reviewers are satisfied with the changes and consider them ready for merging, they approve the pull request.
  - **Second Approach (Merge Commit)**
    - git checkout target-branch
    - git merge source-branch
- **Push the Changes to the Remote Repository**
  ```
  git push origin target-branch
  ```

Pull Operation

- In Git, a "pull" operation is used to fetch changes from a remote repository and integrate them into your local branch.
- This is typically used to update your local repository with changes made by others in the remote repository.
- If you want to pull a specific branch from the remote repository, then you can use command:

```
git pull origin branch-name
```

- You can also fetch changes from the remote repository and merge them into your local branch as separate steps.

```
git fetch
git merge origin/branch-name
```

Available branches on GitHub (remote) Repository

To list the branches available on a remote GitHub repository using the Visual Studio Code terminal, you can use the git ls-remote command.

```
git ls-remote --heads <repository_url>
```

Resolving Merge Conflicts

- Resolving merge conflicts in Git involves addressing conflicts that occur when you're trying to merge one branch into another and there are conflicting changes in the same part of a file.
- When git is unable to automatically resolve conflicts / difference in code of both branches.

Steps to resolve merge conflicts

- Pull Latest Changes
```
git pull origin target-branch
```
- Initiate the Merge
```
git merge source-branch
```
- Identify Conflicted Files
- Resolve Conflicts – Manually
- Add the Resolved Files
```
git add "file name"
```
- Complete the Merge Commit
```
git commit
```
- Push Your Changes
```
git push origin target-branch
```

Undo Operations

Staged Changes

git reset "file-name"	(for a specific file)
git reset	(for all the files)

Commit Changes

git reset HEAD~1	(for previous commit)
git reset "hash-code of commit"	(for a specific commit)
git reset - - hard "hash-code of commit"	

note : last command is used to reset the commit changes along with the code available in VS-code.

Fork Operation

In Git, a "fork" operation refers to the process of creating a personal copy of a repository from another user or organization's repository.
Forking is a common practice on Git hosting platforms like GitHub, GitLab, Bitbucket, etc.

Steps to perform forking:

- Choose a Repository to Fork on GitHub
- Fork the Repository

Now the repository is available for you and you can perform operation according to your requirements or interest.

--homework

Global and Local Configurations

- In Git, you can set user and email configurations globally (for your entire system) or locally (specific to a single Git repository).
- The choice between global and local configurations depends on your needs and the context in which you're working.

Global Configuration (User and Email)

- To update the global Git user and email configuration, you can use the git config command with the --global flag.
- This will set the user and email values that will be used for all Git repositories on your system.
- Here are the steps:
    - Set the Global User Name

      ```
      git config --global user.name "Your Name"
      ```
    - Set the Global Email Address

      ```
      git config --global user.email “Your Email”
      ```
    - Verify Your Changes

      ```
      git config --global –list
      ```

When to use Global Configuration

When You Have Consistent Identity: Use global configuration when you want to maintain a consistent user identity and email address across all Git repositories on your system. It's suitable for your primary or default identity.
Personal Projects: Global settings are ideal for personal projects or repositories where you are the sole contributor, as you don't need to differentiate your identity on a per-repository basis.
Default or Non-sensitive Information: You can safely use global configurations for repositories that don't contain sensitive information, and where it's acceptable to expose your personal identity and email address.

Local Configuration (User and Email)

- To set a local user and email for a specific Git repository, you can use the git config command without the --global flag.
- This will configure the user and email settings only for that particular repository.
- Here are the steps:
    - Set the Global User Name
      ```
      git config user.name "Your Name"
      ```
    - Set the Global Email Address
      ```
      git config user.email "Your Email"
      ```
    - Verify Your Changes
      ```
      git config –list
      ```

These local user and email settings will apply only to the current Git repository. If you work on multiple Git repositories, each one can have its own set of user and email values. Keep in mind that these local settings will take precedence over global settings, which are configured using git config --global.

When to use Local Configuration

Contributing to Different Projects: Use local configuration when you contribute to multiple open-source or collaborative projects. Different projects may have different requirements for user identity and email, so you can set them locally for each repository.
Work-related Repositories: In a work environment, where you contribute to various repositories associated with different projects or teams, local settings allow you to tailor your identity and email according to the project's needs.
Isolating Sensitive Information: Local configurations are helpful when you work on repositories that contain sensitive information, such as work-related projects, and you want to isolate your work-related identity from personal projects.
Testing and Experimentation: Local settings can be useful for testing and experimentation in repositories where you don't want to interfere with your global identity and email.