Version Control

What is Version Control

Version control is a system that manages changes to a file or set of files over time

It allows you to keep track of changes made to your code and collaborate with others on the same codebase

Version control is essential for software development projects and is widely used in the software industry.

Types of Version control systems

There are two main types of version control systems:

1. Centralized version control systems
2. Distributed version control systems

Centralized version control systems (CVCS) store the code in a central repository, and developers make changes to the code directly on the server

Examples of CVCS include CVS and Subversion

Distributed version control systems (DVCS) allow each developer to have their own copy of the code repository, which they can work on independently and then merge changes back to the main repository

Examples of DVCS include Git and Mercurial

NB: In this course we would be focusing on GIT.

How Version Control Works

In version control, changes to the code are stored in a repository. The repository contains all versions of the codebase, along with information about who made each change and when it was made.
When a developer wants to make a change to the code, they create a branch, which is a copy of the codebase. They make their changes on the branch and then merge their changes back into the main codebase when they're ready.
Version control systems also allow developers to work collaboratively on the same codebase. Multiple developers can work on different branches simultaneously, and their changes can be merged together to create a final version of the code.

Benefits of Version Control

- Version control is an essential tool for software development projects and can greatly improve productivity and efficiency.
- Collaboration - Multiple developers can work on the same codebase at the same time, without interfering with each other's work.
- Versioning - Developers can keep track of every change that is made to the code, allowing them to roll back to a previous version if necessary.
- Accountability - Version control systems keep track of who made changes to the code, providing a clear history of who did what and when.
- Backup and recovery - The repository acts as a backup of the code, and allows for easy recovery in case of a system failure, hardware failure or loss of data.
- Experimentation - Developers can create new branches to experiment with new features or ideas, without affecting the main codebase.

Git with Github

GIT

Git is a distributed version control system that is used to manage source code and other files.

It was created by Linus Torvalds in 2005 to manage the development of the Linux kernel.

Git is an open-source project, which means that anyone can contribute to its development.

It is designed to be fast, efficient, and flexible, making it a popular choice for software development teams.

GIT is not github (GitHub is a web-based platform that is used to manage Git repositories.)

Github

GitHub is a web-based platform that is used to manage Git repositories.

It was launched in 2008 and quickly became a popular platform for open-source software development.

GitHub provides tools and features that make it easy for developers to collaborate on code, track changes, and manage project workflows.

It is used by millions of developers around the world for a wide range of software development projects.

Git vs. GitHub: What's the Difference?

Git and GitHub are often used interchangeably, but they are not the same thing. Git is a version control system, while GitHub is a web-based platform that provides hosting for Git repositories. Git can be used without GitHub, but GitHub cannot be used without Git.

Git workflow

- Git workflow is a systematic process that outlines the steps involved in using Git to manage and track changes to code
- Git has a unique workflow that is designed to make it easy for developers to collaborate on code and manage project workflows.
- The basic Git workflow consists of the following stages: clone, branch, add, commit, push, pull request, review, merge.
- Clone: create a local copy of the repository on the developer's machine
- Branch: create a new branch to work on a specific feature or issue
- Add: add changes to the staging area using the 'git add' command
- Commit: create a new commit with a descriptive message using the 'git commit' command
- Push: push the local commits to the remote repository using the 'git push' command
- Pull request: create a pull request to merge changes into the main codebase
- Review: other developers can review the changes, provide feedback, and suggest changes
- Merge: integrate changes made on the developer's branch with the main branch of the codebase using the 'git merge' command.

Setting up a GitHub account

1. Go to the GitHub homepage at https://github.com/.
2. Click on the "Sign up" button at the top right corner of the page.
3. Enter your preferred username, email address, and password in the provided fields.
4. Select the type of account you want to create. You can choose between a free personal account or a paid business account.
5. Choose your plan. If you're signing up for a personal account, you can select the free plan or the paid "Pro" plan.
6. Fill out your personal information and billing details (if applicable) on the next page.
7. Verify your email address by clicking on the link in the email that GitHub sends to you.
8. Congratulations! You now have a GitHub account. You can start creating repositories, collaborating with others, and contributing to open-source projects.

Setting up GitHub on your local system (windows)

1. Download and install Git from the official website https://git-scm.com/download/win.
2. After installing Git, open the Git Bash terminal by right-clicking on a folder or the desktop and selecting "Git Bash Here".
3. Set your username and email by entering the following commands in the Git Bash terminal:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

4. Now you can start using Git and GitHub by cloning repositories, making changes, and pushing them to GitHub.
5. Sign into your github account from your VS Code

Setting up GitHub on your local system (linux)

1. Install Git using “ sudo apt-get install git”
2. Set your username and email by entering the following commands in the terminal:

   ```
   git config --global user.name "Your Name"
   git config –global user.email "youremail@example.com"
   ```

3. Now you can start using Git and GitHub by cloning repositories, making changes, and pushing them to GitHub.
4. Sign into your github account from your VS Code

Take note of these 5 terms

1. Local machine: This refers to the physical computer or device where you are making changes to files in a Git repository.
2. Local repository: This refers to the copy of the Git repository that is on your local machine. It includes all the files, folders, and commit history for the project.
3. Remote repository: This refers to the copy of the Git repository that is hosted on a remote server, such as GitHub or Bitbucket. It is a centralized location where collaborators can push and pull changes from.
4. Staging area: This is a temporary storage area in Git where you can stage changes before committing them to the local repository. The staging area allows you to selectively choose which changes to include in a commit.
5. Branch: A branch in Git is a version of the code that is separate from the main codebase. Branches are often used for feature development or bug fixing, and can be merged back into the main codebase when the changes are complete.

Git in a glance

- Repository: central location to store codebase, create on GitHub by clicking "New Repository"
- Clone: copy repository to local machine to start working
- Commit: create snapshot of code changes using Git CLI
- Push: update repository on GitHub with latest changes using Git CLI
- Branching: create multiple versions of codebase using separate branches
- Create branch: using Git CLI
- Switch branch: using Git CLI
- Merge branches: using Git CLI
- Resolve merge conflicts: using Git CLI
- Collaborate with forks and pull requests: for collaborative development
- Use issues and labels: for organizing and tracking tasks
- Review code with pull requests: tools for code review on GitHub

Basic Git Commands

Git commands for workflow

- Clone: git clone <repository-url> - creates a local copy of the repository on the developer's machine
- Branch: git branch <branch-name> - creates a new branch to work on a specific feature or issue
- Checkout: git checkout <branch-name> - switches to a different branch
- Add: git add <file-name> - adds changes to the staging area
- Commit: git commit -m "commit message" - creates a new commit with a descriptive message
- Push: git push <remote-name> <branch-name> - pushes the local commits to the remote repository
- Pull: git pull - fetches and merges changes from the remote repository to the local repository
- Merge: git merge <branch-name> - integrates changes made on a specific branch with the current branch

Creating a Repository

To create a new Git repository, navigate to the directory where you want the repository to be located and use the command

git init.

Adding files to the staging area

The staging area is where you prepare changes before committing them to your repository. To add files to the staging area, use the following command:

git add <file>

You can also use git add . to add all changes.

Committing changes

Committing changes is the process of saving your changes to the repository. When you commit changes, you should always include a descriptive commit message that explains what you changed. To commit changes, use the following command:

git commit -m "<commit-message>"

Checking the status of a file

Checking the status of a file tells you which files are modified, which files are staged, and which files are not yet tracked by Git. To check the status of files, use the following command:

git status

Viewing differences in two commits

To view the differences between two commits, use the following command:

git diff <commit-hash-1> <commit-hash-2>

Viewing Commit History

Git keeps track of all the commits made to a repository. To view the commit history, use the following command:

git log

Ignoring Files

Sometimes you may have files that you don't want to be tracked by Git, such as log files or configuration files. To ignore files, create a file called .gitignore in the root directory of your repository and list the files to ignore in the file.

For example, to ignore all .log files, add the following line to .gitignore file

*.log

Reverting Changes

If you make a mistake and need to undo a commit, use the following command:

git revert <commit-hash>

Pushing changes to GitHub

Pushing changes sends your local commits to the remote repository on GitHub. To push changes, use the following command:

git push origin <branch-name>

If you are pushing changes for the first time, you will need to use the command:

git push -u origin <branch-name>

Pulling changes from GitHub

Pulling changes gets the latest changes from the remote repository on GitHub and merges them into your local repository. To pull changes, use the following command:

git pull origin <branch-name>

You can include the fetch command

Git fetch && git pull origin <branch-name>

Cloning a Repository

Cloning a repository is the process of creating a local copy of a remote Git repository. To clone a repository, use the following command:

git clone <repository-url>

Branching and Merging

Branching is a feature in Git that allows developers to create multiple versions of their codebase. Each version is stored in a separate branch, which can be merged back into the main codebase when the changes are ready.

Git allows you to work on different features or bug fixes in separate branches. Once a feature is complete, you can merge it back into the main branch. To create a new branch, use the following command:
git branch <branch-name>

To switch to a different branch, use the command:
git checkout <branch-name>

To merge a branch back into the main branch, first switch to the main branch and then use the following command:
git merge <branch-name>

You can also create and switch to a new branch with a single command:
git checkout -b <new-branch-name>

Removing Files

To remove a file from Git and from the file system, use the following command:

git rm <file-name>

To remove a file from Git but keep it in the file system, use the following command:

git rm --cached <file-name>

Renaming Files

To rename a file in Git, use the following command:

git mv <old-file-name> <new-file-name>

Resolving Merge Conflicts

If there are conflicts when merging a branch, Git will pause the merge and ask you to resolve the conflicts manually. To resolve the conflicts, edit the files that have conflicts, stage the changes, and then commit the changes.

Rebasing

Rebasing is a way to change the history of a branch by moving the commits to a new base commit. To rebase a branch, use the following command:

git rebase <new-base-branch>

This will move the commits from the current branch to the new base branch. Rebasing can be useful for keeping the Git history clean and linear.

Tagging

To tag a specific commit with a name, use the following command:

git tag <tag-name> <commit-hash>

Best practices for Git and GitHub:

- Commit frequently in small increments to track changes and avoid conflicts
- Write clear and descriptive commit messages to explain changes
- Use meaningful branch names to understand the purpose of the branch
- Keep repository organized with logical file organization and descriptive names
- Stay up-to-date with changes to the main codebase by pulling frequently
- Carefully review code and provide constructive feedback
- Use Git hooks to automate tasks such as running tests and checking code formatting.

End

Version control

Class Quiz

- Each of you needs to create a GitHub account if you haven't already done so.
- I will create a new repository on GitHub and make a folder for each person. Then, I will add all 18 of you as collaborators.
- Each of you should clone the repository to your local machine using Git terminal.
- Create a new branch in Git with a meaningful name that describes the purpose of the branch. Use the name of your program from the last assignment.
- Make some changes to your folder. Create a file for your assignment in your branch and commit those changes with a clear and descriptive commit message.
- Push your branch to GitHub and create a pull request to merge your changes into the main branch.
- I will review the pull request and provide feedback or approve the changes.
- Once all changes have been reviewed and approved, I will merge the changes into the main branch and score you appropriately (10 marks).