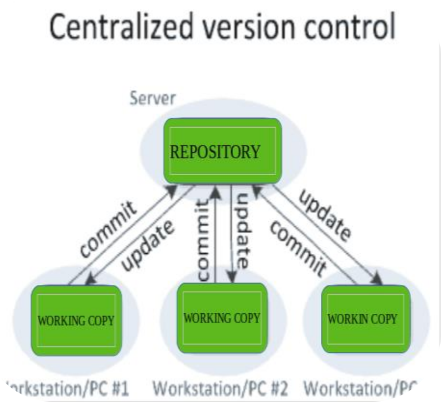
1. What is version control system(VCS)? Explain in your own word.

A Version Control System (VCS) is a software tool that helps developers track and manage changes to files and projects over time. Think of it like a meticulous historian for your code, meticulously recording every modification, who made it, and when, according to Atlassian.

Here's a breakdown of what a VCS is and how it functions:

- Tracking changes: A VCS maintains a history of every change made to a file or a set of files, creating snapshots of each version. This allows you to look back at any point in the project's development and see exactly what it looked like.
- Collaboration: In collaborative environments, multiple developers can work on the same codebase simultaneously without overwriting each other's work. VCS tools help merge these changes and resolve any conflicts that arise.
- Reverting to previous versions: If a mistake is made or a new feature doesn't work as expected, a VCS allows you to easily revert to a previous, stable version of the code. This acts as a safety net, enabling developers to experiment and correct errors without fear of irreversible damage to the codebase.
- Branching and merging: A key feature of VCSs is the ability to create "branches," which are separate lines of development stemming from the main codebase. This allows developers to work on new features or bug fixes in isolation without affecting the main project. Once the work is complete and tested, the changes from the branch can be "merged" back into the main line of development.
- Backup and recovery: VCS acts as a backup system for your code, ensuring that in case of data loss or system failure, you can recover previous versions of your project.

2. What are the two main types of version control systems? Explain with one example each.



The two main types of Version Control Systems (VCS) are:

1. Centralized Version Control Systems (CVCS)

- Explanation: In a CVCS, there's a single, central repository on a server that stores all the versions of the project's files. Developers connect to this central server to "check out" files, make changes locally, and then "commit" their modifications back to the central repository. In this model, everyone works directly with this single source of truth.
- Example: Subversion (SVN) is a well-known example of a CVCS. In an SVN setup, all development activity revolves around a central server, making it easier for smaller teams or projects needing tight control and a single point of reference.

2. Distributed Version Control Systems (DVCS)

- Explanation: In a DVCS, each developer doesn't just check out a copy of the files; they get a complete clone of the entire repository, including its full history, on their local machine. This means they can work offline, commit changes locally, and even create local branches without needing to be constantly connected to a central server. When ready, they can "push" their local changes to a shared, often remote, repository and "pull" updates from other developers.

- Example: Git is the most popular example of a DVCS. With Git, developers have the flexibility to work independently on their local copies and then easily exchange changes with a central repository (like GitHub or GitLab) when they choose.

3. Name any three version control tools and stick or draw their logos:

- **Git**
- **GitHub**
- **GitLab / Bitbucket**

Here are three prominent version control tools and their logos:

1. GitGit is a widely used open-source distributed version control system (DVCS). It is known for its speed, efficiency, and powerful branching and merging capabilities, allowing developers to track changes and collaborate effectively on projects of any size. Git serves as the foundation for other platforms like GitHub, GitLab, and Bitbucket.
2. GitHubGitHub is a web-based hosting service for Git repositories, providing a platform for version control and collaborative software development. It's essentially a cloud-based service built on top of Git, offering a graphical interface and features like pull requests, issue tracking, and code review. GitHub is popular for open-source projects and boasts a large developer community. The platform's mascot, the Octocat, features prominently in its logo.
3. GitLabGitLab is another web-based platform that offers Git repository hosting and integrated DevOps capabilities. Unlike GitHub's primary focus on collaboration, GitLab is designed as an all-in-one DevSecOps platform covering the entire software development lifecycle, including continuous integration and delivery (CI/CD), security testing, and project management. GitLab offers both cloud-based and self-hosted options, catering to various organizational needs. Its logomark is inspired by the Tanuki, a Japanese raccoon dog.

4. What is the difference between Git and GitHub?

S.No.	Git	GitHub
1.	Git is a software.	GitHub is a service.
2.	Git is a command-line tool	GitHub is a graphical user interface
3.	Git is installed locally on the system	GitHub is hosted on the web
4.	Git is maintained by linux.	GitHub is maintained by Microsoft.
5.	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.

S.No.	Git	GitHub
6.	Git is a version control system to manage source code history.	GitHub is a hosting service for Git repositories.
7.	Git was first released in 2005.	GitHub was launched in 2008.
8.	Git has no user management feature.	GitHub has a built-in user management feature.
9.	Git is open-source licensed.	GitHub includes a free-tier and pay-for-use tier.
10.	Git has minimal external tool configuration.	GitHub has an active marketplace for tool integration.
11.	Git provides a Desktop interface named Git Gui.	GitHub provides a Desktop interface named GitHub Desktop.
12.	Git competes with CVS, Subversion, Mercurial, etc.	GitHub competes with GitLab, Bit Bucket, AWS Code Commit, Azure DevOps Server, etc.

5. **Write the steps to create a project folder named VC-Practical-01 and an HTML file index.html inside it using Notepad or VS Code.**

Creating a project folder and an HTML file

Here's how to create the `VC-Practical-01` folder and `index.html` file using both Notepad and Visual Studio Code:

Using Notepad (Windows)

1. Create the Project Folder:
   - Open File Explorer (Windows key + E).
   - Navigate to the location where you want to create your project folder (e.g., your Desktop or Documents folder).
   - Right-click in an empty area.
   - Select New > Folder.
   - Name the folder `VC-Practical-01` and press Enter.
2. Create the `index.html` file:
   - Double-click to open the newly created `VC-Practical-01` folder.
   - Right-click in an empty area inside the folder.
   - Select New > Text Document.
   - Name the file `index.html` (make sure to change the extension from `.txt` to `.html`).
   - Important: Windows might warn you about changing the file extension. Confirm by clicking "Yes."

    - Double-click the `index.html` file to open it in Notepad.
    - Add some basic HTML code to the file. For example:

html

```
<!DOCTYPE html>

<html>

<head>

    <title>My First HTML Page</title>

</head>

<body>

    <h1>Hello from VC-Practical-01!</h1>

    <p>This is my first HTML page.</p>

</body>

</html>
```

Use code with caution.

- Save the file (File > Save or Ctrl + S). You now have a basic HTML page ready!

Using Visual Studio Code (VS Code)

1. Open VS Code and Create/Open the Folder:
    - Open Visual Studio Code.
    - From the top menu, go to File > Open Folder....
    - Navigate to the desired location where you want to create your project folder.
    - Click the New Folder button.
    - Type `VC-Practical-01` and press Enter to create the folder.
    - Select the newly created `VC-Practical-01` folder and click Select Folder (or Open on macOS).
    - If prompted with a "Workspace Trust" dialog, choose to trust the folder.
2. Create the `index.html` file:
    - In the Explorer panel (the sidebar on the left), you'll see your `VC-Practical-01` folder listed.
    - Hover over the `VC-Practical-01` folder name. You'll see several icons appear to the right.
    - Click the New File icon (it looks like a document with a plus sign).
    - Type `index.html` and press Enter. VS Code will automatically open the new `index.html` file in the editor.
    - Add some basic HTML code. VS Code provides excellent features like Emmet to quickly generate boilerplate HTML. You can type `!` and then press Tab to get a basic HTML structure. You can use the following code:

html

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My First HTML Page</title>
</head>
<body>
    <h1>Hello from VC-Practical-01!</h1>
    <p>This is my first HTML page created in VS Code.</p>
</body>
</html>
```

6. Write the HTML code you used in index.html.

```
  <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My First HTML Page</title>
</head>
<body>
    <h1>Hello from VC-Practical-01!</h1>
    <p>This is my first HTML page created in VS Code.</p>
</body>
</html>
```

7. Write down the steps to:

- **Add your project folder into GitHub Desktop**
- **Create a local repository**

Here are the steps to add your existing `VC-Practical-01` project folder into GitHub Desktop and initialize it as a local Git repository:

Step 1: Open GitHub Desktop

1. Launch the GitHub Desktop application on your computer.

Step 2: Add a Local Repository

1. In GitHub Desktop, go to File > Add Local Repository… in the top menu bar.
2. Alternatively, you can drag the `VC-Practical-01` folder directly onto the GitHub Desktop window.

Step 3: Choose the project folder

1. In the "Add Local Repository" dialog box that appears, click the Choose... button.
2. Navigate to the location of your `VC-Practical-01` folder (where you created the `index.html` file).
3. Select the `VC-Practical-01` folder and click Select Folder (or similar button like "Add Repository").

Step 4: Initialize as a Git repository (if not already)

1. If the folder has not yet been initialized as a Git repository (meaning it doesn't have a hidden `.git` folder inside), GitHub Desktop will prompt you, saying something like "This directory does not appear to be a Git repository. Would you like to create a repository here instead?".
2. Click Create a repository (or a similar prompt button).
3. Ensure the name is correct: The suggested repository name should be `VC-Practical-01`, matching your folder name. If not, correct it.
4. Optional: Add a .gitignore and License:
    - .gitignore: You can select a .gitignore template based on your project's language (e.g., HTML if there's a relevant one, or just a basic one for now) to prevent certain files from being tracked by Git. For simple HTML projects, this might not be critical initially.
    - License: You can choose a license if you plan to share your project publicly. This is optional.
5. Click Create repository.

Step 5: Initial Commit

1. GitHub Desktop will now recognize the files in your `VC-Practical-01` folder as changes that need to be committed to the new local repository.
2. In the "Summary (required)" field, type a concise commit message (e.g., "Initial commit for VC-Practical-01 project").
3. Optionally, you can add a longer description in the "Description" field.
4. Click the Commit to main (or `Commit to master`, depending on your default branch name) button.

8. What is a commit? o Write the step or syntax you used while making a commit in GitHub Desktop. o After committing and pushing, were any changes visible on GitHub? If yes, where?

What is a Commit?

A **commit** in Git is a snapshot of your project at a specific point in time. It records changes made to files in your repository, along with a message describing the changes. Commits help track the history of your project and enable collaboration with others.

🛠 Syntax / Step to Commit in GitHub Desktop:

Here are the steps to make a commit using GitHub Desktop:

1. **Open GitHub Desktop** and select your repository.
2. Make sure your changes are staged (they will be listed under "Changes").
3. In the bottom-left panel:
    - Enter a **summary** for your commit (required).
    - Optionally, enter a **description**.
4. Click **"Commit to main"** (or your current branch name).
5. Then click **"Push origin"** to push your changes to GitHub.

🌐 After Committing and Pushing: Were Any Changes Visible on GitHub?

✅ **Yes**, changes were visible on GitHub:

- Your latest commit appears in the **"Commits"** tab of your repository.
- The **files** that were changed will reflect the updates under the **"Code"** tab.
- You can also see the commit message and details by clicking on the specific commit under **"History"** in GitHub Desktop or GitHub.com.

9. What steps did you follow to publish your project to GitHub using GitHub Desktop?

✅ **Steps to Publish Your Project to GitHub Using GitHub Desktop**

Here's a step-by-step guide to publishing a local project to GitHub using **GitHub Desktop**:

🔧 **Step 1: Open GitHub Desktop**

- Launch **GitHub Desktop** on your computer.

💼 **Step 2: Add Your Local Repository (if it's not already there)**

If you already have a local project folder:

1. Go to `File` > `Add Local Repository`.
2. Browse and select the folder containing your project.
3. Click **Add Repository**.

📤 **Step 3: Publish the Repository to GitHub**

Once your project is added to GitHub Desktop:

1. At the top bar, click on **"Publish repository"**.
2. In the popup window:
   - **Name** your repository.
   - Optionally add a **description**.
   - Choose whether to make it **Public** or **Private**.
   - Make sure the checkbox for **"Push to GitHub"** is selected.
3. Click **"Publish Repository"**.

10. Write the code you added when you created about.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>About Us</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 40px;
      background-color: #f9f9f9;
      color: #333;
    }
    h1 {
      color: #007acc;
    }
    p {
      font-size: 1.1em;
      line-height: 1.6;
    }
  </style>
</head>
<body>
  <h1>About Us</h1>
    <p>Welcome to our website! We are passionate about creating quality                webexperiences
  and sharing knowledge.</p>
 <p>This page gives you insight into our team, mission, and values.</p>
</body>
</html>
```

11. What changes did you make in index.html after your first commit?

Great question! Here's an example of the kinds of changes someone might make to `index.html` after their first commit:

Example Changes Made to `index.html` After First Commit

1. **Added a header section** with a site title and navigation.
2. **Included a welcome message** or introduction paragraph.
3. **Added some basic styling** (either inline or linked CSS).
4. **Fixed typos or formatting** from the initial commit.

Sample Description of Changes:

- Added `<header>` with the website name and navigation links.
- Included a main `<section>` with a welcome message.
- Improved page structure by adding semantic HTML tags like `<main>` and `<footer>`.
- Updated the title tag for better SEO.
- Added comments for clarity.

Sample code snippet showing a change:

Before (initial commit):

```
<!DOCTYPE html>
<html>
<head>
  <title>My Website</title>
</head>
<body>
  Hello world!
</body>
</html>
```

After (second commit):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>My Website - Home</title>
</head>
<body>
  <header>
    <h1>Welcome to My Website</h1>
    <nav>
      <a href="index.html">Home</a> |
      <a href="about.html">About</a>
    </nav>
  </header>
  <main>
    <p>Hello world! This is my updated homepage with better structure.</p>
  </main>
  <footer>
    <p>© 2025 My Website</p>
  </footer>
</body>
</html>
```