Introduction to GitHub

What is GitHub?

GitHub is a web-based platform that uses Git for version control. It provides a suite of tools and features to facilitate software development, especially in collaborative environments.

The primary functions and features of GitHub include:

- **Repositories**: Central storage locations for project files.
- **Version Control**: Tracks and manages changes to code.
- **Branching and Merging**: Facilitates parallel development and integration.
- **Pull Requests**: Allows contributors to propose changes and review code before merging.
- **Issues and Project Management**: Tools for tracking tasks, bugs, and project progress.
- **Continuous Integration/Continuous Deployment (CI/CD)**: Automated testing and deployment pipelines.
- **Collaboration Tools**: Wikis, discussions, and code reviews.

GitHub supports collaborative software development by allowing multiple developers to work on the same project simultaneously. It ensures that changes are tracked and merged systematically, reducing conflicts and maintaining code integrity.

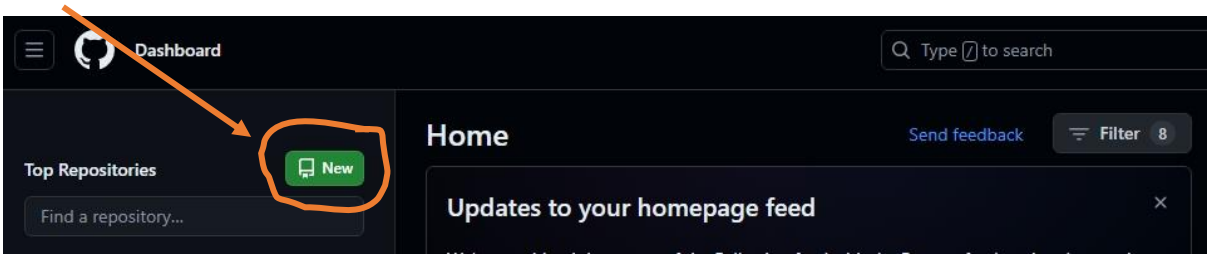
Repositories on GitHub

What is a GitHub repository?

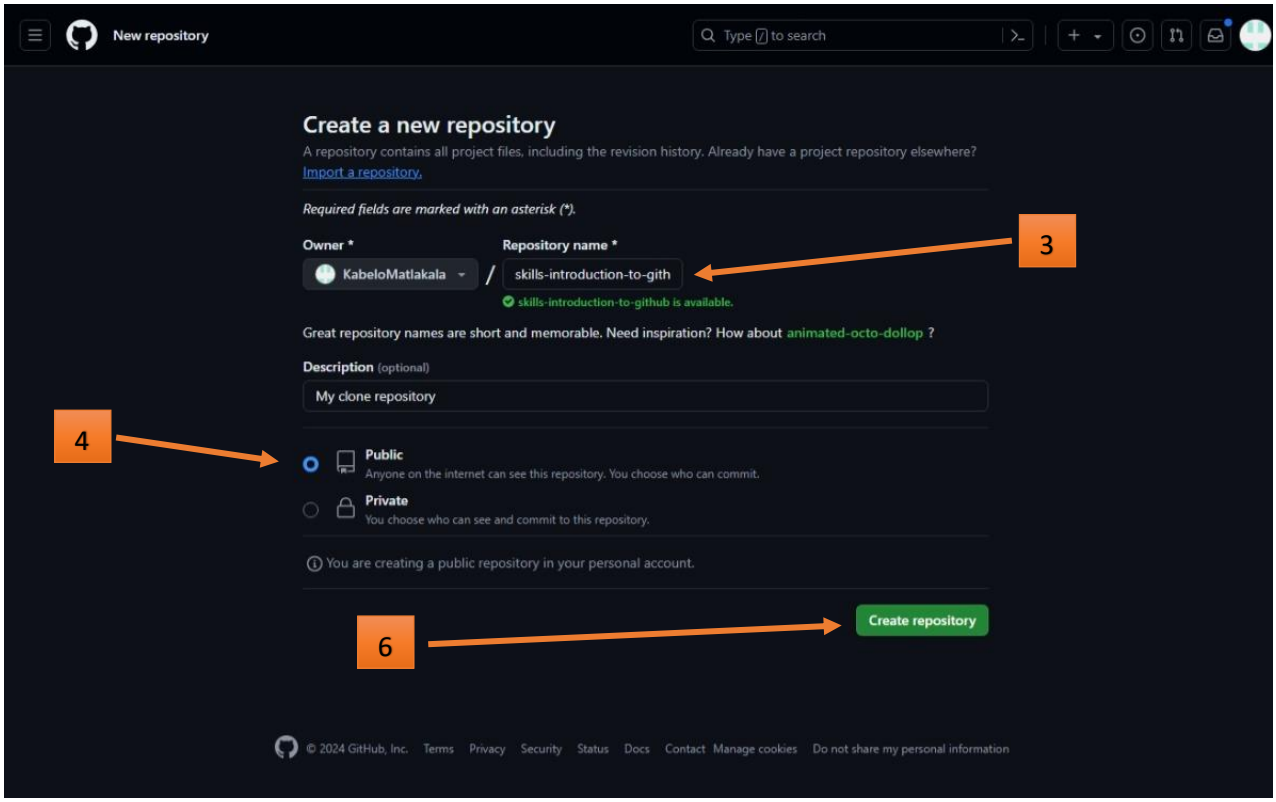
A **GitHub repository (repo)** is a central location where all the files and history of a project are stored. Repositories can be public or private, and they enable version control and collaboration.

To create a new repository:

1. Sign in to GitHub.
2. Click the "New" button next to the repository list on your dashboard or go to https://github.com/new.



3. Fill in the repository name and optional description.
4. Choose between public or private visibility.
5. Optionally, initialize the repository with a README file, .gitignore file, or license.
6. Click "Create repository."



Essential elements of a repository include:

- **README**: A markdown file explaining the project's purpose, setup instructions, and usage.
- **.gitignore**: Specifies which files and directories Git should ignore.
- **LICENSE**: Defines the terms under which the project's code can be used and shared.
- **Source Code**: The actual code files and directories.
- **Documentation**: Guides and reference materials for users and developers.

Version Control with Git:

Concept of version control in the context of Git.

Version Control is the process of tracking and managing changes to software code.

Git is a distributed version control system that allows multiple developers to work on a project concurrently without interfering with each other's work.

Key features of Git include:

- **Commit History**: A record of all changes made to the codebase.
- **Branches**: Separate lines of development that can later be merged.
- **Merging**: Combining changes from different branches.
- **Rebasing**: Reapplying commits on top of another base commit.

GitHub enhances version control by providing a remote, centralized platform for storing Git repositories.

It offers additional tools for collaboration, such as:

- **Pull Requests**: Facilitate code reviews and discussions before changes are merged.
- **Protected Branches**: Prevent direct pushes to critical branches to ensure all changes go through review.
- **Web Interface**: Easy navigation of the repository, commit history, and branches.
- **Integration with CI/CD**: Automatically test and deploy code changes.

Branching and Merging in GitHub

What are branches in GitHub, and why are they important?

Branches in GitHub are parallel versions of a repository, allowing multiple lines of development to occur simultaneously.

They are important because they:

- Enable feature development and bug fixes without disrupting the main codebase.
- Facilitate experimentation and isolated work until changes are ready to be merged.
- Allow different developers or teams to work on separate tasks concurrently.

Process of creating a branch, making changes, and merging it back:

1. **Create a Branch**:

```sh
git checkout -b new-feature
```

Or on GitHub, use the web interface to create a new branch from the repository's branch list.

2. **Make Changes**: Edit files, add new files, and commit changes to the new branch:

```sh
git add .
git commit -m "Added new feature"
```

3. **Push the Branch to GitHub**:

```sh
git push origin new-feature
```

4. **Create a Pull Request**:
    - Go to the GitHub repository.
    - Click the "Compare & pull request" button next to your branch.
    - Review changes and submit the pull request for review.
5. **Merge the Branch**:
    - Once the pull request is approved, click the "Merge pull request" button.
    - Alternatively, use the command line to merge:

```sh
git checkout main
git merge new-feature
```

6. **Delete the Branch**:

```sh
git branch -d new-feature
git push origin --delete new-feature
```

Pull Requests and Code Reviews:

What is a pull request in GitHub, and how does it facilitate code reviews and collaboration?

A pull request (PR) is a request to merge changes from one branch into another.

It facilitates code reviews and collaboration by:

- Allowing team members to review and discuss changes before merging.
- Ensuring code quality and adherence to standards.
- Providing a history of proposed changes and their discussions.

Steps to create and review a pull request:

1. **Create a Pull Request**:
    - Push your branch to GitHub.
    - Navigate to the repository on GitHub.
    - Click the "Compare & pull request" button.
    - Fill in the title and description of the pull request.
    - Assign reviewers and add any relevant labels.
    - Click "Create pull request."
2. **Review a Pull Request**:
    - Go to the "Pull requests" tab in the repository.
    - Select the pull request you want to review.
    - Examine the code changes, leave comments, and request changes if necessary.
    - Once satisfied, approve the pull request.
3. **Merge the Pull Request**:
    - After approval, the pull request can be merged using the "Merge pull request" button.
    - Choose the appropriate merge method (merge commit, squash, or rebase).
    - Confirm the merge.

GitHub Actions:

What are GitHub Actions and how they can be used to automate workflows.

GitHub Actions is an automation tool integrated into GitHub that allows developers to create custom workflows for their projects. These workflows can be triggered by various events (e.g., pushes, pull requests) and can perform tasks such as running tests, building code, and deploying applications.

Example of a simple CI/CD pipeline using GitHub Actions:

1. **Create a Workflow File**: In your repository, create a directory named `.github/workflows` and add a file named `ci.yml`.
2. **Define the Workflow**:

```yaml
name: CI

on: [push, pull_request]

jobs:
        build:
          runs-on: ubuntu-latest

          steps:
            - name: Checkout code
              uses: actions/checkout@v2

            - name: Set up Node.js
              uses: actions/setup-node@v2
              with:
                node-version: '14'

            - name: Install dependencies
              run: npm install

            - name: Run tests
              run: npm test
```

This workflow triggers on every push or pull request, checks out the code, sets up Node.js, installs dependencies, and runs tests.

Introduction to Visual Studio

What is Visual Studio?

Visual Studio is an integrated development environment (IDE) from Microsoft used for developing applications across various platforms, including Windows, web, cloud, and mobile.

Key features include:

- **Code Editor**: Advanced editing features with IntelliSense.
- **Debugger**: Powerful debugging tools with breakpoints, watches, and call stack inspection.
- **Designer**: Visual designers for building user interfaces.
- **Integrated Tools**: Built-in support for version control, databases, testing, and deployment.
- **Extensions**: Support for a wide range of plugins and extensions.

Visual Studio differs from Visual Studio Code (VS Code) as follows:

- **Visual Studio**: Full-featured IDE with comprehensive tools for large-scale software development.
- **Visual Studio Code**: Lightweight, cross-platform code editor focused on simplicity and speed, suitable for quick development tasks and scripting.

Integrating GitHub with Visual Studio:

1. **Install GitHub Extension** (if not already installed):
   - Open Visual Studio.
   - Go to `Extensions > Manage Extensions`.
   - Search for "GitHub" and install the GitHub Extension for Visual Studio.
2. **Clone a Repository**:
   - Go to `File > Open > Open from Source Control`.
   - Select `GitHub`.
   - Sign in to your GitHub account.
   - Select the repository you want to clone and click `Clone`.
3. **Work on the Repository**:
   - Use Visual Studio's tools to edit, commit, and push changes.
   - Access Git features directly within Visual Studio (e.g., branching, merging, pull requests).

Integration enhances the development workflow by providing:

- **Seamless Version Control**: Direct access to Git features without leaving the IDE.
- **Enhanced Collaboration**: Easy creation and review of pull requests within Visual Studio.
- **Integrated Tools**: Unified environment for coding, debugging, testing, and version control.

Debugging in Visual Studio:

Visual Studio offers a robust set of debugging tools, including:

- **Breakpoints**: Pause execution at specific lines of code to examine state.
- **Step Through Code**: Step into, over, or out of code lines to follow execution flow.
- **Watch Windows**: Monitor the values of variables and expressions.
- **Immediate Window**: Execute code and evaluate expressions during debugging.
- **Call Stack**: View the sequence of function calls leading to the current point.
- **Exception Handling**: Catch and handle exceptions to understand issues.

Developers can use these tools to identify and fix issues by:

1. **Setting Breakpoints**: Identify where issues may occur and pause execution to inspect state.
2. **Stepping Through Code**: Follow the code execution flow to understand logic and find errors.
3. **Using Watch Windows**: Monitor variable values and ensure they are as expected.
4. **Examining the Call Stack**: Understand the context of the current execution point and trace back to the source of issues.
5. **Handling Exceptions**: Catch and analyse exceptions to prevent crashes and improve error handling.

Collaborative Development using GitHub and Visual Studio:

GitHub and Visual Studio can be used together to support collaborative development by combining GitHub's version control and collaboration features with Visual Studio's powerful development tools. This integration streamlines workflows, improves code quality, and enhances team productivity.

Example: Open-Source Project Consider an open-source project where multiple contributors work on adding features, fixing bugs, and improving documentation.

The project benefits from GitHub and Visual Studio integration as follows:

1. **Repository Management**: The project repository is hosted on GitHub, providing version control, issue tracking, and project management tools.
2. **Collaborative Development**: Contributors clone the repository into Visual Studio, where they can edit code, debug, and test changes.
3. **Pull Requests**: Contributors create pull requests from their feature branches, facilitating code reviews and discussions.
4. **Automated Workflows**: GitHub Actions automate testing and deployment, ensuring code quality and speeding up release cycles.
5. **Debugging and Testing**: Visual Studio’s debugging tools help contributors identify and fix issues before submitting pull requests.

This integration enhances collaboration by providing a seamless development environment, making it easier for contributors to focus on writing and improving code while leveraging GitHub's collaboration and automation features.