Introduction to Full Stack Java Script

Git and GitHub

* Git and GitHub are two closely related tools that are widely used in software development for version control and collaboration.
* While Git is a distributed version control system (DVCS),
* GitHub is a web-based platform built around Git for hosting repositories and facilitating collaboration among developers.

Prerequites:

* Git installation, Git Command Line Interface/Shell – Bash, Github account , any Editor

Git:

* <https://git-scm.com/>
* Installing Git Bash (Bourne Again Shell) provides all the necessary emulation to access all the command line tools Git needs.

Here are some of the key features of Git and GitHub:

1. Distributed Version Control: Git is a distributed version control system, which

means that every user has a full copy of the repository, including its history. This allows for easy branching, merging, and offline work.

2. Branching and Merging: Git allows developers to create multiple branches to work on different features or fixes independently. Branches can be merged back into the main branch (typically `master` or `main`), allowing for seamless integration of changes.

3. History Tracking: Git tracks changes to files over time, allowing developers to view the history of changes, who made them, and when they were made. This facilitates debugging, auditing, and understanding of project evolution.

4. Collaboration: Git enables collaboration among developers by allowing them to clone repositories, work on changes locally, and push their changes to a shared repository. It also supports workflows such as forking, pull requests, and code reviews.

5. Staging Area: Git uses a staging area (also known as the index) to prepare changes before committing them to the repository. This allows developers to selectively stage changes and create more granular commits.

6. Performance: Git is designed to be fast, even with large repositories and extensive history. It employs various optimizations to ensure efficient operations, such as delta compression and packfiles.

GitHub:

1. Remote Hosting: GitHub provides a web-based platform for hosting Git repositories. Developers can push their local repositories to GitHub, making them accessible to collaborators from anywhere with an internet connection.

2. Issue Tracking: GitHub includes an issue tracking system that allows users to create, assign, and track issues and bugs. Issues can be linked to commits and pull requests, providing context and facilitating collaboration.

3. Pull Requests: GitHub's pull request feature enables developers to propose changes to a repository and request that they be reviewed and merged. Pull requests can include discussion, comments, and automated checks, making them a central part of the collaboration workflow.

4. Code Review: GitHub provides tools for code review, including line comments, diff views, and inline discussions. This allows collaborators to provide feedback on proposed changes before they are merged into the main branch.

5. Integration with Third-Party Services: GitHub integrates with a wide range of third-party services and tools, such as continuous integration (CI) systems, project management platforms, and code analysis tools. This enables developers to automate workflows and streamline development processes.

6. Wikis and Documentation: GitHub repositories can include wikis and documentation, making it easy to maintain project-related information, guidelines, and resources alongside the codebase.

7. Community and Social Features: GitHub fosters a large community of developers and open-source projects.

Users can follow projects, star repositories, contribute to open-source projects, and discover new projects through GitHub's social features.

Overall, Git and GitHub together provide a powerful set of tools for version control, collaboration, and project management, making them essential components of modern software development workflows.

**React Js**

React.js, often referred to simply as React, is an open-source JavaScript library developed by Facebook for building user interfaces (UIs) and single-page applications (SPAs).

(GoogleMaps,Gmail,youtube,twitter,Facebook,github,Netflex-Uses SPA)

It is widely used for creating interactive and dynamic web applications with a focus on component-based architecture and reusability.

Here are some key aspects and features of React.js:

# Component-Based Architecture:

1. Components: React.js is built around the concept of reusable components. A component in React is a self-contained unit that encapsulates a piece of UI and its behavior. Components can be composed together to build complex UIs.

2. Component Lifecycle: React components have a lifecycle with various methods that allow developers to hook into different stages of a component's existence, such as mounting, updating, and unmounting. This enables developers to perform actions like fetching data, updating the UI, or cleaning up resources at specific points in the component's lifecycle.

Virtual DOM:

1. Virtual DOM: React uses a virtual DOM (Document Object Model) to efficiently update the UI.
2. Instead of directly manipulating the browser's DOM, React creates a lightweight, in-memory representation of the DOM structure. When the state of a component changes, React compares the virtual DOM with the previous version and computes the minimum number of DOM operations needed to update the actual DOM, resulting in better performance.

2. Reconciliation: React's diffing algorithm, also known as reconciliation, optimizes the process of updating the DOM by minimizing unnecessary updates and re-renders. This helps improve the overall performance and responsiveness of React applications.

**JSX (JavaScript XML):**

1. JSX Syntax: React uses JSX, a syntax extension for JavaScript that allows developers to write HTML-like code directly within JavaScript. JSX simplifies the process of creating and composing UI components by providing a familiar and expressive syntax.

2. Component Rendering: JSX allows developers to define the structure and appearance of UI components declaratively, making it easier to visualize and understand the component hierarchy and behavior.

Unidirectional Data Flow:

1. State Management: React follows a unidirectional data flow model, where data flows from parent components to child components via props (properties). Components can maintain internal state using React's state mechanism, allowing them to manage and respond to changes independently.

2. Immutable Data: React encourages the use of immutable data structures and pure functions to update state and props. This helps prevent unintended side effects and makes it easier to reason about the behavior of components.

# React Hooks:

1. Functional Components: React introduced Hooks, a feature that allows developers to use state and other React features in functional components without needing to write class components. Hooks enable functional components to manage state, perform side effects, and tap into the React lifecycle.

2. useState, useEffect, useContext: Some of the commonly used React Hooks include useState for managing component state, useEffect for performing side effects, and useContext for accessing context within functional components.

# Eco-System and Tooling:

1. React Router: React Router is a popular library for routing in React applications. It provides a declarative way to define navigation paths and render different components based on the current URL.

2. Redux, MobX: Libraries like Redux and MobX are commonly used with React for managing global application state and handling complex state management scenarios.

3. React Native: React Native is a framework for building cross-platform mobile applications using React. It allows developers to write mobile applications using React and JavaScript, targeting iOS, Android, and other platforms.

React.js continues to evolve with updates, improvements, and an active community of developers contributing to its ecosystem. It has become a cornerstone technology for building modern web applications and is widely adopted by organizations and developers worldwide.

Angular Js

AngularJS is a JavaScript framework developed and maintained by Google.

It is used for building dynamic web applications, particularly single-page applications (SPAs), where content is dynamically loaded and updated on a single HTML page without the need for full page reloads.

Here are some key features and aspects of AngularJS:

Features of AngularJS:

1. Two-Way Data Binding: AngularJS features two-way data binding, which means that changes to the model (data) automatically update the view (UI) and vice versa. This simplifies the synchronization of data between the model and the view, reducing the need for manual DOM manipulation.

2. Directives: Directives are a core feature of AngularJS that allow developers to extend HTML with custom attributes and behaviors. Directives enable the creation of reusable components and the addition of dynamic functionality to HTML elements.

3. Dependency Injection: AngularJS provides a built-in dependency injection mechanism that helps manage dependencies between different components of an application. Dependency injection promotes modularity, testability, and maintainability by decoupling components and making them easier to manage and test in isolation.

4. MVC Architecture: AngularJS follows the Model-View-Controller (MVC) architectural pattern, where the application logic (controller), data (model), and presentation (view) are separated into distinct components. This separation of concerns makes it easier to develop, test, and maintain complex web applications.

5. Templates and Expressions: AngularJS uses HTML templates with embedded expressions, allowing developers to bind data and logic directly to the UI. Expressions are evaluated in the context of the current scope, enabling dynamic rendering of content based on data and user interactions.

6. Routing: AngularJS includes a powerful routing module that enables developers to define routes, map them to controllers and views, and navigate between different parts of the application. Routing in AngularJS facilitates the creation of single-page applications with multiple views and navigation paths.

7. Form Validation: AngularJS provides built-in support for form validation, allowing developers to define validation rules, display validation messages, and handle user input validation. AngularJS's form validation features help ensure data integrity and provide a better user experience.

8. Testing Support: AngularJS includes features and utilities for testing applications, including unit testing and end-to-end testing capabilities. The framework is designed to be testable, with support for mocking dependencies, injecting test data, and automating test execution.

9. Built-in Services: AngularJS provides a set of built-in services that offer common functionality such as HTTP communication, data manipulation, localization, and logging. These services can be easily injected into controllers, directives, and other components to perform specific tasks within the application.

10. Community and Ecosystem: AngularJS has a large and active community of developers, contributors, and third-party libraries that extend its capabilities and provide additional features and tools for building web applications.

AngularJS has been widely adopted for building dynamic and interactive web applications, and its successor, Angular (also known as Angular 2+), continues to evolve with new features, improvements, and best practices for modern web development.

MongoDB

MongoDB is a popular open-source NoSQL database management system known for its flexibility, scalability, and ease of use. It is designed to store and manage large volumes of data, including unstructured and semi-structured data, making it well-suited for modern web applications, analytics, and other data-intensive use cases. Here are some of the salient features of MongoDB:

1. Document-Oriented:

MongoDB is a document-oriented database, which means it stores data in flexible, JSON-like documents. Each document can have its own structure and schema, allowing for dynamic and schema-less data modeling. This flexibility makes MongoDB well-suited for storing and managing diverse types of data.

2. Scalability:

MongoDB is designed to scale horizontally across multiple servers and clusters to handle large volumes of data and high throughput workloads. It supports automatic sharding, which distributes data across multiple nodes in a cluster, enabling horizontal scaling without sacrificing performance.

3. High Performance:

MongoDB is optimized for high performance and low latency. It uses memory-mapped files and native data structures to store and retrieve data efficiently. MongoDB's query optimizer and indexing capabilities help optimize query performance, making it suitable for real-time analytics and transactional workloads.

4. Flexible Data Model:

MongoDB's flexible data model allows developers to store data in a way that closely mirrors the structure of their applications. Documents can contain nested fields, arrays, and other complex data types, enabling rich data representations and reducing the need for complex joins and relationships.

5. Rich Query Language:

MongoDB provides a powerful query language with support for a wide range of operations, including CRUD (Create, Read, Update, Delete) operations, aggregation, text search, geospatial queries, and more. MongoDB's query language is expressive and flexible, allowing developers to retrieve and manipulate data in a variety of ways.

6. Indexing and Aggregation:

MongoDB supports indexing and aggregation capabilities to improve query performance and enable advanced data analysis. Developers can create indexes on fields to accelerate query execution and enforce unique constraints. MongoDB's aggregation framework provides a powerful set of operators and stages for data aggregation, grouping, and transformation.

7. Replication and High Availability:

MongoDB supports replica sets, which are self-healing, distributed clusters that provide high availability and data redundancy. Replica sets consist of multiple nodes, including primary and secondary nodes, which replicate data asynchronously to ensure fault tolerance and data durability.

8. Automatic Failover:

MongoDB's replica sets feature automatic failover, which means that in the event of a primary node failure, one of the secondary nodes is automatically elected as the new primary node. This ensures continuous availability and minimal downtime in the event of hardware failures or network partitions.

9. Security:

MongoDB provides robust security features to protect data at rest and in transit. It supports authentication, role-based access control (RBAC), encryption, and auditing capabilities to secure sensitive data and comply with regulatory requirements.

10. Community and Ecosystem:

MongoDB has a large and active community of developers, contributors, and users who contribute to its development, documentation, and ecosystem. MongoDB offers comprehensive documentation, tutorials, and resources to help developers get started and leverage its features effectively.

Overall, MongoDB's flexibility, scalability, performance, and rich feature set make it a popular choice for a wide range of applications, from small-scale projects to large-scale enterprise deployments.

**Node Js**

Node.js is a powerful, open-source, server-side JavaScript runtime environment built on **Chrome's V8 JavaScript engine.** It enables developers to build scalable and high-performance network applications using JavaScript. Here are some of the salient features of Node.js:

1. Asynchronous and Event-Driven:

Node.js operates on a non-blocking, asynchronous model, which allows it to handle concurrent connections and I/O-bound operations efficiently. It uses an event-driven architecture and callbacks to handle asynchronous tasks, making it well-suited for building real-time applications, streaming services, and APIs that require high concurrency.

2. Single-Threaded, Non-Blocking I/O Model:

Node.js uses a single-threaded event loop to handle multiple concurrent connections without blocking the execution of other tasks. It employs non-blocking I/O operations, such as file system operations and network requests, to maximize throughput and minimize latency, resulting in high performance and responsiveness.

3. JavaScript Everywhere:

Node.js enables developers to use JavaScript for both client-side and server-side development, providing a unified language and ecosystem for building full-stack applications. Developers can share code between the client and server, reuse libraries, and leverage their existing JavaScript skills across different environments.

4. npm (Node Package Manager):

Node.js comes with npm, the largest ecosystem of open-source libraries and modules for JavaScript development. npm provides access to thousands of reusable packages that can be easily installed, managed, and integrated into Node.js applications using a simple command-line interface.

5. Lightweight and Scalable:

Node.js is lightweight and resource-efficient, making it suitable for building scalable and high-performance applications, including microservices, APIs, and real-time systems. Its modular architecture and small footprint make it easy to deploy and manage in production environments, even on distributed systems and cloud platforms.

6. Cross-Platform Compatibility:

Node.js is cross-platform and runs on various operating systems, including Windows, macOS, and Linux. This enables developers to write and deploy Node.js applications on different environments without modification, enhancing portability and interoperability across platforms.

7. Built-in HTTP and Networking Modules:

Node.js includes built-in modules for handling HTTP requests, creating web servers, and performing network operations. Developers can build custom web servers, RESTful APIs, and WebSocket services using Node.js's native HTTP and networking capabilities, without relying on external libraries or frameworks.

8. Community and Support:

Node.js has a vibrant and active community of developers, contributors, and users who contribute to its development, documentation, and ecosystem. The Node.js community provides comprehensive documentation, tutorials, and resources, as well as support through forums, chat channels, and online communities.

9. Tools and Libraries:

Node.js has a rich ecosystem of tools, frameworks, and libraries that extend its functionality and streamline development workflows. From web frameworks like Express.js and Koa.js to testing frameworks like Mocha and Jest, developers have access to a wide range of tools and libraries to build, test, and deploy Node.js applications.

10. Microservices Architecture:

Node.js is well-suited for building microservices-based architectures, where applications are composed of small, independent services that communicate over the network. Its lightweight and scalable nature, combined with its support for asynchronous and event-driven programming, make it an ideal platform for developing microservices-based applications.

Overall, Node.js's asynchronous, event-driven architecture, JavaScript-centric approach, vibrant ecosystem, and cross-platform compatibility make it a popular choice for building modern web applications, APIs, and microservices.

**Express JS**

Express.js is a minimalist web application framework for Node.js, designed to simplify the process of building web applications and APIs.

It provides a robust set of features and middleware that enable developers to create scalable and efficient server-side applications using JavaScript. Here are some of the salient features of Express.js:

1. Web Application Framework:

Express.js is a lightweight and flexible web application framework that runs on top of Node.js. It provides a simple and intuitive API for defining routes, handling HTTP requests and responses, and managing middleware.

2. Middleware:

Express.js uses middleware to handle tasks such as parsing request bodies, serving static files, authentication, logging, and error handling. Middleware functions are executed sequentially in the order they are defined, allowing developers to modularize application logic and extend the functionality of Express.js easily.

3. Routing:

Express.js provides a powerful routing system that allows developers to define routes for handling different HTTP methods (GET, POST, PUT, DELETE, etc.) and URL patterns. Routes can contain route parameters, query parameters, and optional parameters, making it easy to extract data from incoming requests and respond accordingly.

4. Template Engines:

Express.js supports various template engines, such as Pug (formerly Jade), EJS, Handlebars, and Mustache, for rendering dynamic HTML content on the server side. Template engines allow developers to generate HTML dynamically based on data from the server and pass it to the client for rendering.

5. Middleware and Error Handling:

Express.js provides built-in middleware for common tasks such as parsing JSON and URL-encoded data, handling cookies and sessions, and compressing responses. Additionally, Express.js allows developers to define custom middleware functions for performing application-specific tasks and error handling.

6. Static File Serving:

Express.js allows developers to serve static files, such as HTML, CSS, JavaScript, images, and other assets, directly from the file system or a specified directory. This simplifies the process of serving client-side resources and improves application performance by offloading static file serving to the web server.

7. RESTful APIs:

Express.js is commonly used for building RESTful APIs (Representational State Transfer) that adhere to REST principles. It provides features such as route handlers, middleware, and response formatting that make it easy to design and implement RESTful API endpoints for client-server communication.

8. Integration with Database:

Express.js integrates seamlessly with various databases and ORMs (Object-Relational Mappers) such as MongoDB, Mongoose, MySQL, PostgreSQL, Sequelize, and SQLite. Developers can use middleware and libraries to connect to databases, perform CRUD operations, and handle database transactions efficiently.

9. Scalability and Performance:

Express.js is designed to be lightweight, fast, and scalable, making it suitable for building high-performance web applications and APIs. Its asynchronous and non-blocking I/O model, combined with Node.js's event-driven architecture, allows Express.js applications to handle large numbers of concurrent connections and requests efficiently.

10. Community and Ecosystem:

Express.js has a large and active community of developers, contributors, and users who contribute to its development, documentation, and ecosystem. The Express.js community provides comprehensive documentation, tutorials, examples, and third-party middleware and plugins to extend the functionality of Express.js and streamline development workflows.

Overall, Express.js's simplicity, flexibility, extensibility, and performance make it a popular choice for building web applications, APIs, microservices, and server-side applications using Node.js and JavaScript.

Amos q Haviv, MEAN WEB DEVELOPMENT 2ND ED

VASAN SUBRAMANIAM PRO MERN STACK ,FULL STACK WE APP DEV

**Git and Github**

## 1. Step 0: Install git and create a GitHub account

a) download and Install git

<https://git-scm.com/downloads>

**clone for latest version :**

git clone https://github.com/git/git

git bash

b) create github account

c) any editor

* git and GitHub are **not**the same thing.
* Git is an open-source, version control tool created in 2005 by developers working on the Linux operating system;
* GitHub is a company founded in 2008 that makes tools which integrate with git.
* Web based platform for hosting git repository.
* You do not need GitHub to use git, but you cannot use GitHub without using git.
* Alternatives to GitHub, such as GitLab, BitBucket, and “host-your-own” solutions such as gogs and gittea.

## Step 1: Create a local git repository

* create a new [repository](https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository) (or often, 'repo', for short).
* Create and Navigate to the project folder
* To initialize a git repository in the root of the folder, run the [git init](http://git-scm.com/docs/git-init) command:

## Step 2: Add a new file to the repo

* Add new files to the project folder , modify them or delete content.
* Git is aware about the changes but untrack ,means git wont track the changes to the repo until we explictly till it.
* After creating the new file, you can use the [git status](http://git-scm.com/docs/git-status) command to see which files git knows exist.

## Step 3: Add a file to the staging environment

#### Set The staging environment and then do commit the changes

* A [commit](https://docs.github.com/en/free-pro-team@latest/github/getting-started-with-github/github-glossary#:~:text=the%20repository%20owner.-,commit,who%20made%20them%20and%20when.) is a record of what changes you have made since the last time you made a commit.
* To add a file to a commit, you first need to add it to the staging environment using
* Git add <filename> command
* Git add -A or git add -all
* Staging environment is also called staging or index.
* Git status

## Step 4: Create a commit

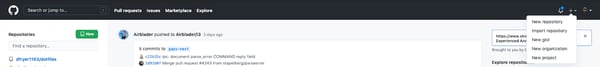
Use git commit -m “message”

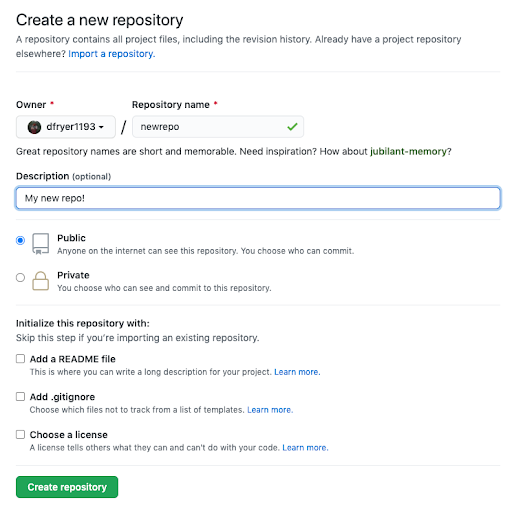
Git status

## Step 5: Create a new branch

* **Branches allow you to work on different parts of a project without impacting the main branch.**
* When the work is complete, a branch can be merged with the main project.
* You can even switch between branches and work on different projects without them interfering with each other.
* **Branches allow you to move back and forth between 'states' of a project.**
* One can make a new feature without making changes to the main project while developing it. This is where [git branches](https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell) come in.
* Branching in Git is very lightweight and fast!
* Official git docs describe branches this way: ‘A branch in Git is simply a lightweight movable pointer to one of these commits.’
* can [merge](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging) your changes from your branch into the primary branch.
* Run [git checkout -b <my branch name>](http://git-scm.com/docs/git-checkout).
* git branch
* git merge <my branch name>

## Step 6: Create a new repository on GitHub

* To create a new repo on GitHub, log in and go to the GitHub home page. You can find the “New repository” option under the “+” sign next to your profile picture, in the top right corner of the navbar:



* GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally.
* push an existing repository from the command line' section:

mnelson:myproject mnelson$ git remote add origin <https://github.com/cubeton/mynewrepository.git>

mnelson:myproject mnelson$ git push -u origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 263 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/cubeton/mynewrepository.git

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

### [view raw](https://gist.github.com/cubeton/3a2616c44e35ca68a6b0/raw/41e5758cfdbd7db8a1659c1adaba9346680097f9/addgithub.md)[addgithub.md](https://gist.github.com/cubeton/3a2616c44e35ca68a6b0#file-addgithub-md)hosted with ❤ by [GitHub](https://github.com/)

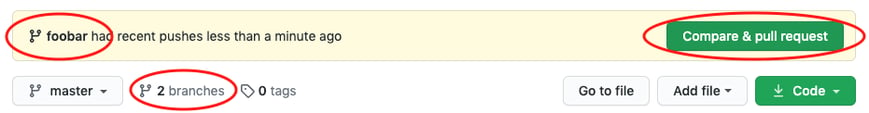
git remote add origin URL specifies that you are adding a remote repository, with the specified URL, as an origin to your local Git repo.

$ git remote add origin https://github.com/avinashvagmare/project\_demog.git

## Step 7: Push a branch to GitHub

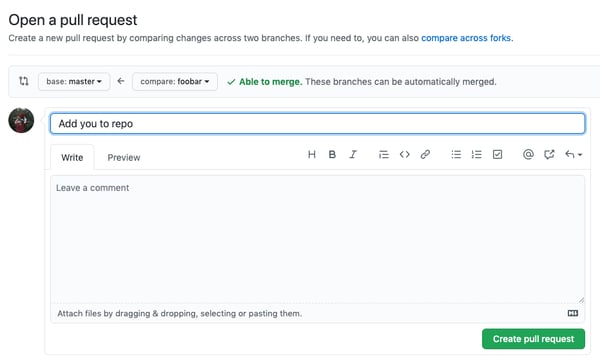
Run the below command

mnelson:myproject mnelson$ git push origin my-new-branch

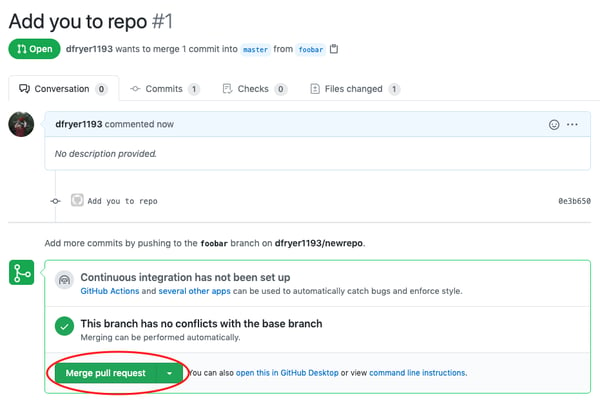


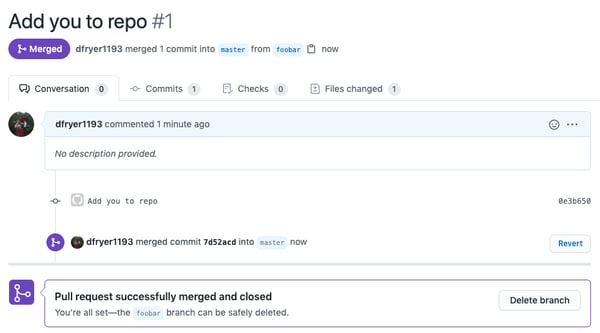
## Step 8: Create a pull request (PR)

* A pull request (or PR) is a way to alert a repo's owners that you want to make some changes to their code. It allows them to review the code and make sure it looks good before putting your changes on the primary branch.



## Step 9: Merge a PR





## Step 10: Get changes on GitHub back to your computer

* mnelson:myproject mnelson$ git pull origin master
* Now we can use the [git log](http://git-scm.com/docs/git-log) command again to see all new commits.

Pushing to local repository:

git remote add origin <https://github.com/avinashvagmare/hello-world.git>

### [Propose changes to someone else's project](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/fork-a-repo#propose-changes-to-someone-elses-project) ( How to create pull request from github)

For example, you can use forks to propose changes related to fixing a bug. Rather than logging an issue for a bug you have found, you can:

* Fork the repository.
* Make the fix.
* Submit a pull request to the project owner.

### [Use someone else's project as a starting point for your own idea.](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/fork-a-repo#use-someone-elses-project-as-a-starting-point-for-your-own-idea)

**Fork a repository**

1. A fork is copy of the original repository.it enables us to freely experiment with changes without affecting the original project.

A fork is a new repository that shares code and visibility settings with the original “upstream” repository.

By convention, your forked repository is called the origin repository, while the original repository is the upstream repository. We can use these aliases to differentiate them.

Example : https://github.com/kedark3/demo

ii) clone the repository

Cloning a repo means creating a copy of a repository in your local environment. You should clone your forked repository when contributing to an open-source project.

git clone <copied-url>

**git clone** https://github.com/<github-username>/<repository-name>.git

## iii) navigate to the project's directory

### example

cd kedar\_demo

kedar\_demo $ git status

\*creating new branch under kedar\_demo and move to it

kedar\_demo $ git branch branch\_kedar\_demo

git checkout branch\_kedar\_demo

## iv)Add Changes to the Staging Area

### Add Single or Multiple Files to the Staging Area

Commit the changes

## Synchronize Changes

When you are working on changes, there is a possibility that the main branch on the upstream repository has already merged in some pull requests. So, the state of the origin and your local repositories at this time will no longer be the same as the upstream.

### Updating the **origin** Repository

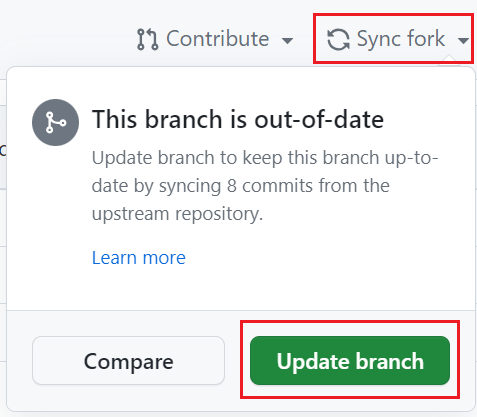
First, go to the origin repository on GitHub to check if it's up to date with the upstream.

You can push your changes when no change exists in the upstream repository.

If you get a message indicating a branch in the origin repository is eight commits behind the upstream repository on GitHub

To update the origin repository:

1. Click the Sync fork dropdown button.
2. Click the green Update branch button.



### Pull Changes

Now that your origin is up to date with the upstream repository, it's time to pull the changes and update your local one.

Pull is a way of getting new changes from the remote to the local repository.

To pull the changes, ensure that you are on your working branch. You can do so by running git status:

$git pull origin main

Now You can now push your changes if you don't need to resolve conflicts.

## Push Changes

Now, it's time to push your changes. This means moving changes from the local to the remote repository.

You always want to push your changes to the origin repository. To do that, run this command in your terminal:

git push origin <branch-name>

git push -u origin <branch-name>

Revert and Reset command in git

The purpose of the git revert command is to remove all the changes a single commit made to your source code repository.

When you revert a Git commit, the changes from the targeted commit are removed from your local workspace.

A new commit is also created to reflect the new state of your repository.

When you git revert a commit, only the changes associated with that commit are undone. Cumulative changes from subsequent commits aren't affected. If you wish to undo every change since a given commit occurred, you'd want to issue a [hard git reset](https://www.theserverside.com/video/How-to-use-the-git-reset-hard-command-to-change-a-commit-history), not revert.

To [undo every change](https://www.theserverside.com/video/Dont-git-revert-that-last-commit-git-reset-instead) that has happened since a given commit occurred, use git reset.

Source for pushing on to github repo https://www.jcchouinard.com/add-a-file-to-github-with-git-bash/

The proper way to push a new project into an existing GitHub repository follows these steps:

1. Create a GitHub repository for the existing project.
2. Copy the GitHub URL for the new repo to the clipboard.
3. Perform a git init command in the root folder of the existing project.
4. Add all of the existing project’s files to the Git index and then commit.
5. Add the GitHub repo as a remote reference for the existing project.
6. Perform a git push operation with the -u and -f switches.
7. Verify that the existing project’s files have been pushed to GitHub.

Another approach :

To quickly add your project to an existing GitHub repo, just follow these steps:

1. Create a new GitHub repo that contains a README file.
2. Use Git to clone the GitHub repo locally.
3. Copy your project files into the folder created by the clone.
4. Perform a git add . and a git commit.
5. Push your changes up to GitHub.

How to resolve Merge Conflict on GitHub

Step1: Assuming local repository head ->Master

//Create two Feature Branchs and make changes to same file in two branches and push the commit to github for merge using pull request.

$ git branch b2

$ git branch b3

$ git checkout b2

Step: Make changes to file two.txt and commit the changes from b2

$ git add . && git commit -m “line added -feature1-branch b2”

$git checkout b3

Make changes to file two.txt and commit the changes

$ git checkout b3

$ git add . && git commit -m “line added -feature 2 -branch b3”

Step : pushing the commit from Branch b2

$ git checkout b2

$ git push origin b2

$ git checkout b3

$ git push origin b3

Step : create pull request from origin/master to b2

Create pull request from origin/master to b3

Pull request # 2

Pull request # 3

Step: Look for merging pull request from branch b2 to origin/master

Click pull request for branch b2

Merge and confirm merge

Look for merging pull request from branch b3 to origin/master

Click pull request for branch b3

When try to Merge, Conflict files is displayed

Click the web editor or command line

Edit the changes : make changes /retain changes from b2 /b3

Then Mark as resolved

Then Merge.

Origin/Master is updated.

Step: pulling the updates from github to local repository master

$ git fetch

$ git log --all --graph

$ git checkout master

$ git pull origin master

Updates / Merges are available to master on local repository

Deleting the branch :

Command: $ git branch -D b2

$ git branch -D b3

**How to create a pull request in Github (detail procedure with example)**

fork a repo, make changes, and ask the maintainers to review and merge it.

if you want to contribute to a project, the simplest way is to:

1. Find a open source project you want to contribute to
2. Fork it.
3. Clone it to your local system
4. Make a new branch
5. Make your changes
6. Push it back to your repo
7. Click the **Compare & pull request** button
8. Click **Create pull request** to open a new pull request

Detail procedure for creating a pull request:

Click the fork button to fork the repo you want to contribute.

This creates a new copy of my demo repo under your GitHub user account with a URL like:

https://github.com/<YourUserName>/demo

The copy includes all the code, branches, and commits from the original repo.

Next, clone the repo by opening the terminal on your computer and running the command:

git clone https://github.com/<YourUserName>/demo

Once the repo is cloned, you need to do two things:

1. Create a new branch by issuing the command:

git checkout -b new\_branch

1. Create a new remote for the upstream repo with the command:

git remote add upstream https://github.com/kedark3/demo

In this case, "upstream repo" refers to the original repo you created your fork from.

Now you can make changes to the code. The following code creates a new branch, makes an arbitrary change, and pushes it to **new\_branch**:

$ git checkout -b new\_branch

Switched to a new branch ‘new\_branch’

$ echo “some test file” > test

$ cat test

Some test file

$ git status

On branch new\_branch

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

test

nothing added to commit but untracked files present (use "git add" to track)

$ git add test

$ git commit -S -m "Adding a test file to new\_branch"

[new\_branch (root-commit) 4265ec8] Adding a test file to new\_branch

1 file changed, 1 insertion(+)

create mode 100644 test

$ git push -u origin new\_branch

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 918 bytes | 918.00 KiB/s, done.

Total 3 (delta 0), reused 0 (delta 0)

Remote: Create a pull request for ‘new\_branch’ on GitHub by visiting:

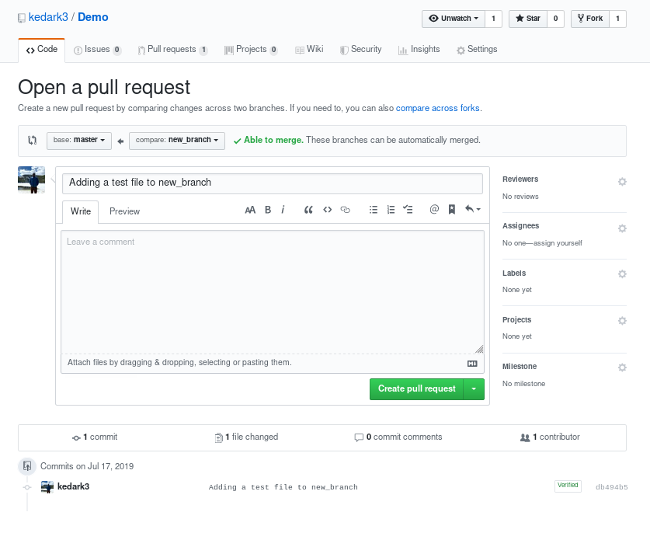
Remote: http://github.com/example/Demo/pull/new/new\_branch

Remote:

\* [new branch] new\_branch -> new\_branch

Once you push the changes to your repo, the **Compare & pull request** button will appear in GitHub.

Click it and you'll be taken to this screen:



Open a pull request by clicking the **Create pull request** button. This allows the repo's maintainers to review your contribution. From here, they can merge it if it is good, or they may ask you to make some changes.

<https://opensource.com/article/19/7/create-pull-request-github>

<https://github.com/kedark3/Demo/pull/1609>

$ git branch branch\_kedar\_demo

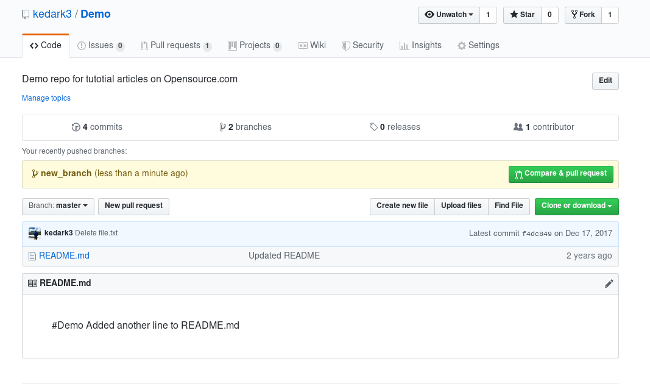
$ git checkout branch\_kedar\_demo

Modify some files like test and test1

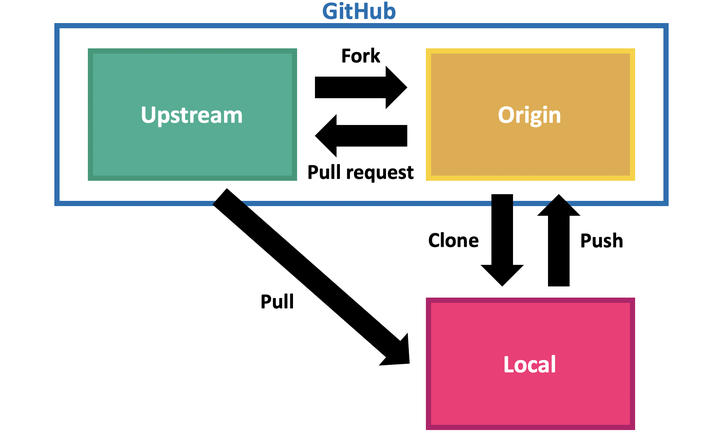
$ git pull origin master

$ git remote -v

$ git push -u origin branch\_kedar\_demo



Fork and Pull request workflow related commands



The workflow comprises the following steps which are described in more detail in the subsequent sections:

1. Fork a GitHub repository: navigate to a repository on GitHub and click the Fork button.
2. Clone the repository locally:

git clone https://github.com/yourusername/repo.git.

1. Add remote called “upstream” pointing to the original repository: git remote add upstream <https://github.com/original_user/repo.git>.

#check for remotes

git remote -v

1. Checkout a new branch (here called “new\_feature”):

git checkout -b new\_feature

git branch -a

1. Make desired changes to the local repository on this branch.
2. Pull new changes from remote:

git checkout master,

 git pull upstream master.

go back to your branch and merge it with the master to incorporate any new changes:

Sync dev branch: git checkout new\_feature,

git merge master.

1. Push changes to your remote repository:

git push origin new\_feature.

1. Open a pull request on GitHub merging your changes with the upstream (original) repository.

To open a pull request, go to the GitHub website, navigate to your “new\_feature” branch and follow the prompts to open a pull request.

1. Once the pull request is accepted, you’ll want to pull those changes into your origin (forked repository).

Change to master: git checkout master and pull: git pull upstream master.

1. Delete your feature branch using the GitHub website or, delete the local branch: git branch -d new\_feature, and delete the remote: git push origin --delete new\_feature.

https://www.tomasbeuzen.com/post/git-fork-branch-pull/