What is Open Source Software?

Open source software is software with source code that anyone can inspect, modify, and enhance.

Famous Examples

1. Android -> Linux
2. Mozilla
3. Apache
4. Python
5. LibreOffice
6. VLC media player

Open source is not only software!!!

Closed Source/Proprietary Software?

Proprietary software is software that is deemed to be non-free and its creator exercises a legal monopoly with the help of copyright and intellectual property law to restrict the user from freely sharing the software or modifying it.

Famous Examples

1. Microsoft Office
2. Adobe Photoshop
3. Oracle

Open Source Vs Proprietary Software

1. Source Code
2. Price
3. Development Process
4. Service Component
5. Usability
6. Security

Why you should contribute to Open Source?

1. Gain Industry Experience
2. Learning
3. Profile Building
4. Networking
5. People Skills
6. Joy of giving

How you can contribute to an Open Source Project

Coding based Contribution

1. Adding new features
2. Bug-fixing
3. Automate Project setup
4. Improve tooling and testing
5. Review others code

Non-coding based Contributions

1. Writing Documentation
2. Writing Tutorials
3. Mentoring Contributors
4. Answering Projects
5. Organize events/promotions

Required Skill for Open Source Contribution

Tangible Skills

Programming

Version Control

Tech stack

Non-tangible skills

Hacker Mindset

Thick skinned

People skills

Roles in an Open Source Project

Author

The person/s or organization that created the project

Owner

The owner has administrative ownership of the project

Maintainers

These people are responsible for driving the vision and goals of the project

Contributors

Contributors add to the project in one way or another

Community Members/Users

People who use the project. They might be active in conversations or express their opinion on the project's direction

Anatomy of an Open Source Project

Licence

If a project does not have an open-source license, then it is not open source. The license helps protect contributors and users. Businesses and savvy developers usually won't touch a project without this protection.

Contributing

These are guidelines that help people who contribute to the project know exactly what is expected from them

Documentation

A detailed description of the code in the software

Code of Conduct

A written set of rules on how everyone should behave while collaborating on the project

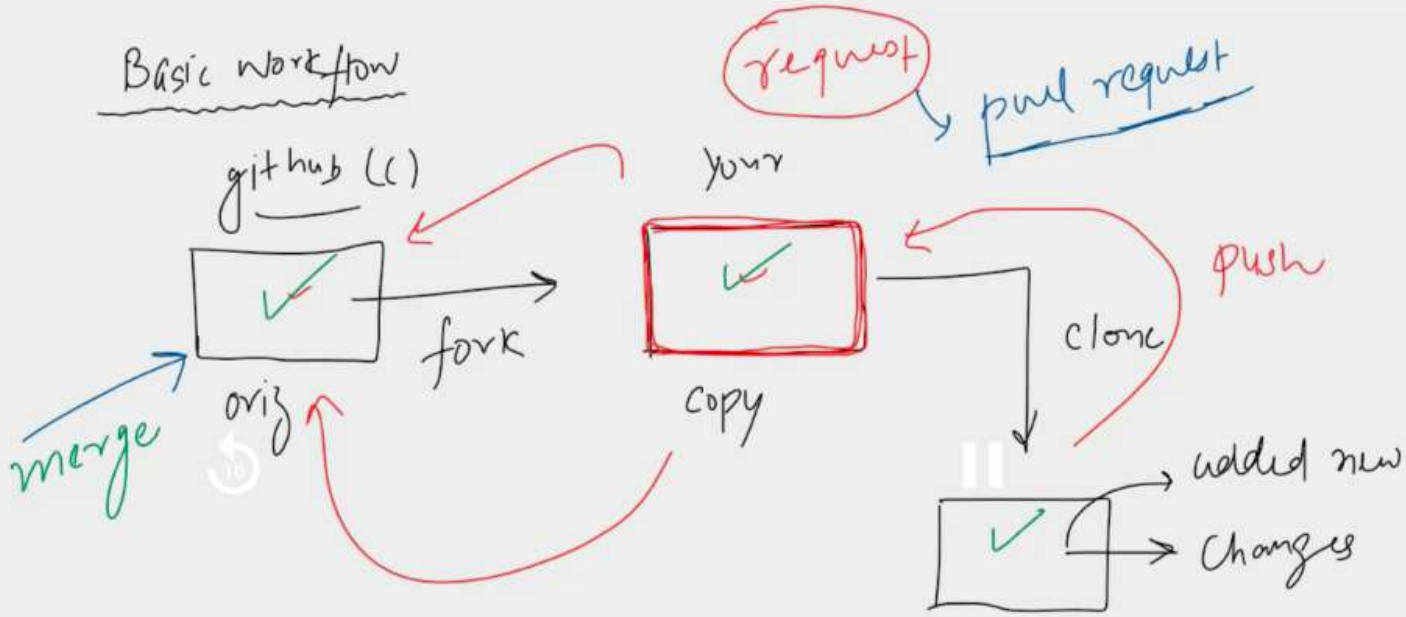
Readme

This is a manual that explains how to get started with a project.

OneNote for Windows 10

Home Insert Draw View Help

Shapes Ink to Shape Ink to Text



learnwith.campusx.in
j.s.aniket@gmail.com
+918180865500

00:04:38 / 01:09:37

GitHub issues is a interface for asking questions about the project to a project maintainer in an open way that lets everyone see what's being done with the project.

"issues" doesn't mean that there's actually a bug, it can just be any change that needs to be made to the project.

Each issue can:

- have a label or multiple labels applied to it
- can be assigned to an individual
- can be assigned a milestone (for example the issue will be resolved by the next major release)

One of the most important aspects of the issue tracker is that each issue can have its own comments, so a conversation can form around the issue.

Another thing that's nice about issues is:

- they let you subscribe to an issue so you'll be notified of new comments and code changes
- you can communicate back and forth with a project maintainer on a specific change

GitHub issues support Markdown.

You can also create your own issues. Make sure you write good descriptive issues.

Next, it's a good idea to look at the GitHub issues for the project

- look at the existing issues to see if one is similar to the change you want to contribute
- if necessary create a new issue
- communicate the changes you'd like to make to the project maintainer in the issue

When you start developing, commit all of your work on a topic branch:

- do not work on the master branch
- make sure to give the topic branch clear, descriptive name

As a general best practice for writing commits:

- make frequent, smaller commits
- use clear and descriptive commit messages
- update the README file, if necessary