# The Power of Git/GitHub/GitLab

# The Need for Organization

- Every project needs organization. It prevents unnecessary mistakes, minimizes confusion, and increases productivity.

- Programmers organize their projects by using something called version control systems.

- These systems make it so a group of programmers can work on the same software without the confusion of making sure everyone is working on up to date code.

# The Need for Organization

- It allows programmers to edit the same files as well, and doesn’t overwrite the progress of another team member.

- Without version control, it is similar to making an entire project in Word. You would need to save and send the document to other team members every time you make a change.

- Version control systems act somewhat like Microsoft’s OneDrive, where the team saves their code and everyone can get the updated version.

# What is Git?

- Git is one of the most commonly used version control systems.
- From the creators, “Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.”
- Git uses different commands, such as stage, commit, pull, push, etc to organize a project. We will go more in depth later.

# Why use Git?

- Git helps keep projects organized, as well as providing a way to look back at the history of your code if you want to undo changes or recover lost code
- It prevents members of a project from accidentally overwriting another member's work without saving their changes
- It keeps a single up-to-date version of the project, so there is no confusion as to which version is the most updated
- Team members can work on different subversions of a project known as "branches" in which they can implement unfinished code without it affecting the whole project

# Why use Git?


Companies & Projects Using Git
Google
Microsoft
LinkedIn
NETFLIX
PostgreSQL
RAILS
Qt
GNOME
eclipse

# Cloning

- Using the “clone” function is the first step of working on a project, and it allows you to download a copy of an entire project, and initialize git to track changes made in that folder and connects it with the repository.

- This allows you to immediately begin working without worrying about setting up git and manually connecting it to a repository.

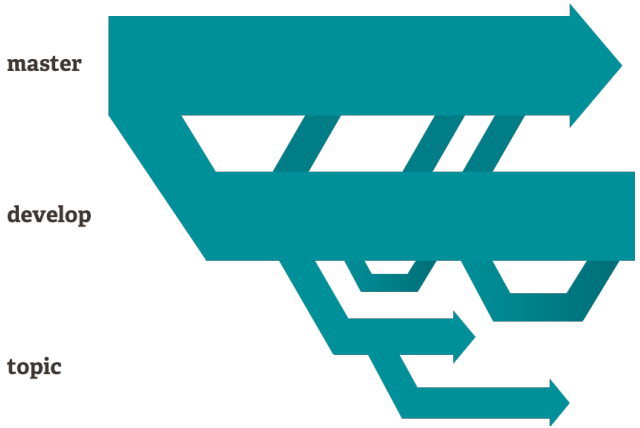
# Branching and Merging

- Branches are used to work on a project without affecting the main code by committing their changes to different, separate versions of the codebase.

- This allows someone to write unfinished or unpolished code on a separate branch from the main codebase (which is on it’s own branch named “main”) and test out some ideas without breaking the main code.

# Branching and Merging

- You can break projects up into pieces like: Production, testing, and branches for normal day to day work.
- New features could also be in another branch. This allows you to switch in between developing the new feature and working on the main branch seamlessly.
- If the new feature or experimentation does not end up working you can just delete it without issues.

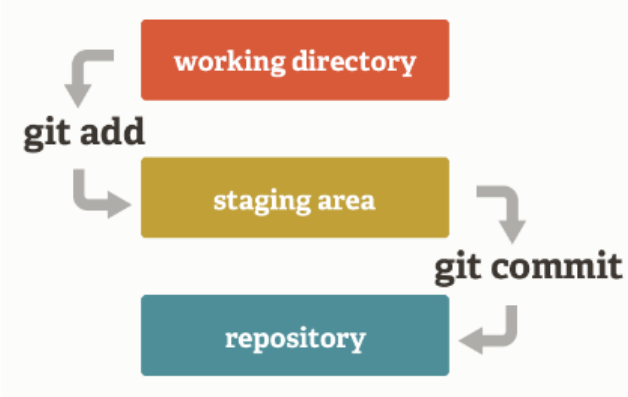


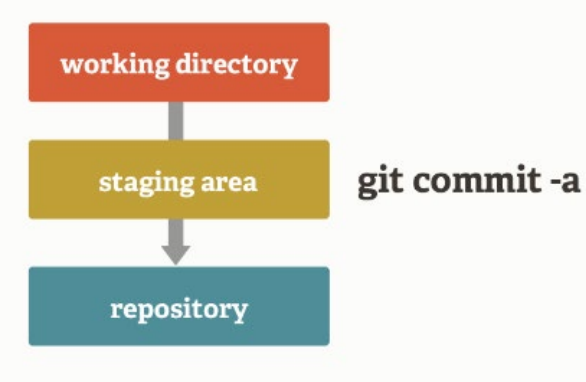
# Commits

- “Commits” are snapshots of different versions of a project’s code.
- Every time a project member commits their changes, they first “stage” what changes they wish to upload, and then write a message that describes these updates.
- These messages help keep an organized log of what has been done on a project.
- After creating a commit, the changes are still not seen by any other project members. The “push” command will take care of this.

# Staging Area

- Git has something called a staging area. This is a intermediate area where commits can be reviewed and edit them before completing the commit.
- You can stage some or all of your change files making it easy to group logically similar changes into multiple commits.
- This functionality is not required though. You can just directly commit all changes to the main code.



# Push/pull commands

- After making a commit, the user's changes are still only seen by themselves and not other members of the project.
- In order to share the changes, the user must "push" their changes to the repository.
- For other members to receive their changes on their own machine, they must "pull" the changes from the repository, which will download any commits from other members onto their computer.

# Advantages of Git

Now that you understand the basics of Git, what makes Git superior to alternatives?

- Small and Fast
- Distributed
- Data Assurance

# Small and Fast

- Git performs all of its operations locally giving it an advantage on systems that communicate with servers for project management.
- Git was written in C to be fast and designed from the beginning to handle large projects, such as the Linux Kernel, which uses Git and is available on GitHub.
- Although Git is slower on the initial clone because it downloads the entire history of the project along with the current version, when compared to Subversion, another version control system, is Git is one or two magnitudes faster.

# Distributed

- The first step to working on a project is cloning the project.
- Because this is the first step, every worker has a full backup of the project. If the main project is lost, there are many backups ready for use.
- Many different styles of workflow arise from the easy branching. Such as: Subversion-Style Workflow, Integration Manager Workflow, Dictator and Lieutenants Workflow.
- These different styles are a topic for another presentation but I wanted them included for those who are interested.

# Data Assurance

- Git uses a data model that ensures the integrity of the project.
- Git checks the “integrity” of every file that is put it in and again when you take it out. This means you can’t get anything out of Git that was not put in.
- It is not possible to change the file, date, commit message, or any other data without changing the ID of everything after it, making it impossible to hide a change from others.
- If you have a commit ID, you can see if the project or any of its history has been changed since that last commit.

# Sum Git Up

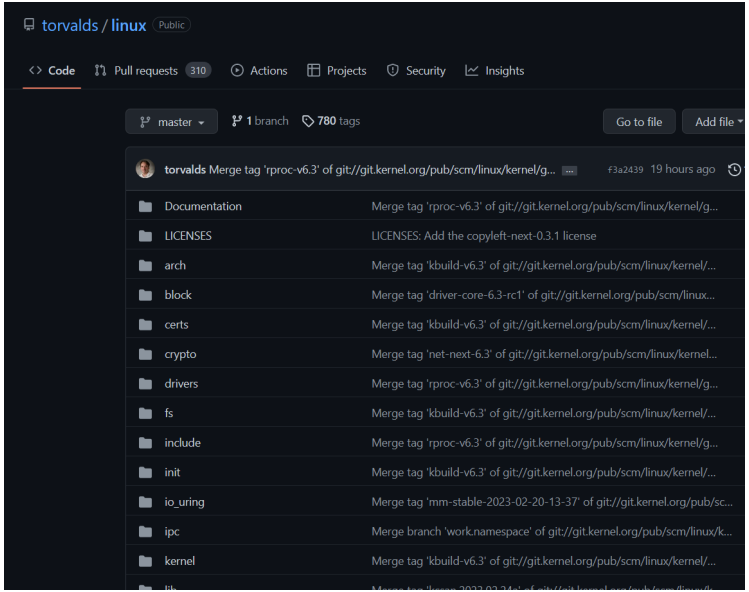
- Git is controlled by many different commands, like **clone, commit, stage, push, pull**
- You can split the project into different **branches**
- Git was designed to be **small and fast**
- Next to impossible to lose and entire project because the project is **distributed**
- You can easily see if the project has been changed with its **data assurance**

# What is GitHub?

- GitHub is a web based software development management tool and code repository
- GitHub provides hosting for your project's code, as well as a way to visualize changes, report bugs and issues, feature requests, wikis, etc.
- It can also be used to find other projects, or recruit others to help work on your own project.

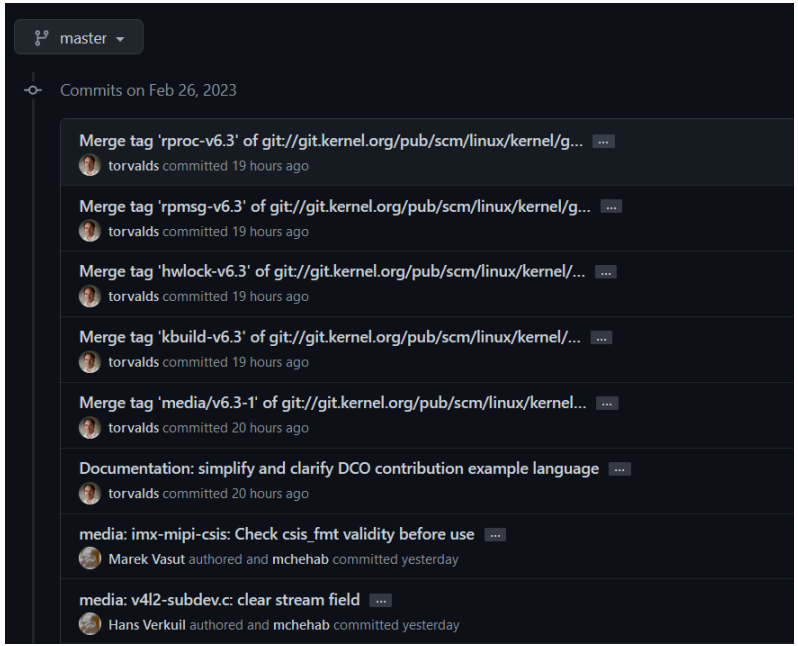
# Project Files

- GitHub's project files are typically organized within a repository.
- A Repository is a collection of files, folders, and metadata that make up a project.
- The repository can be thought of as a container that holds all the project files, including source code, documentation, images, and other assets.
- GitHub repositories can be either public or private



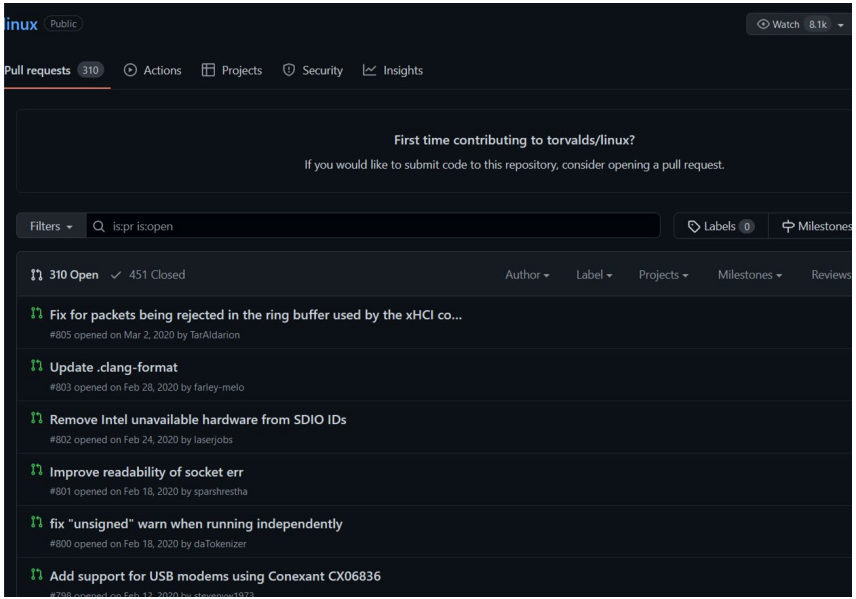
# Commit history

- Commit history refers to the record of changes that have been made to a code repository over time
- When a developer makes changes to the code, they can create a new commit that records those changes.
- It includes information about who made the commit, when it was made, and a description of the changes that were made.



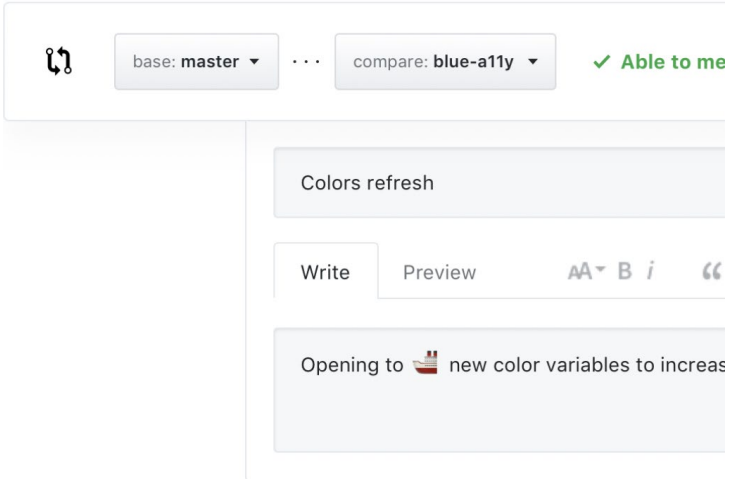
# Pull Requests

- Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. This would be done when you want to merge a branch that you have made changes on.
- Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow -up commits before your changes are merged into the base branch.
- Start a new feature or propose a change to existing code with a pull request —a base for your team to coordinate details and refine your changes.



# Code Review

- On GitHub, lightweight code review tools are built into every pull request.
- Your team can create review processes that improve the quality of your code and fit neatly into your workflow.
- Pull requests are fundamental to how teams review and improve code on GitHub.
- Evolve projects, propose new features, and discuss implementation details before changing your source code.

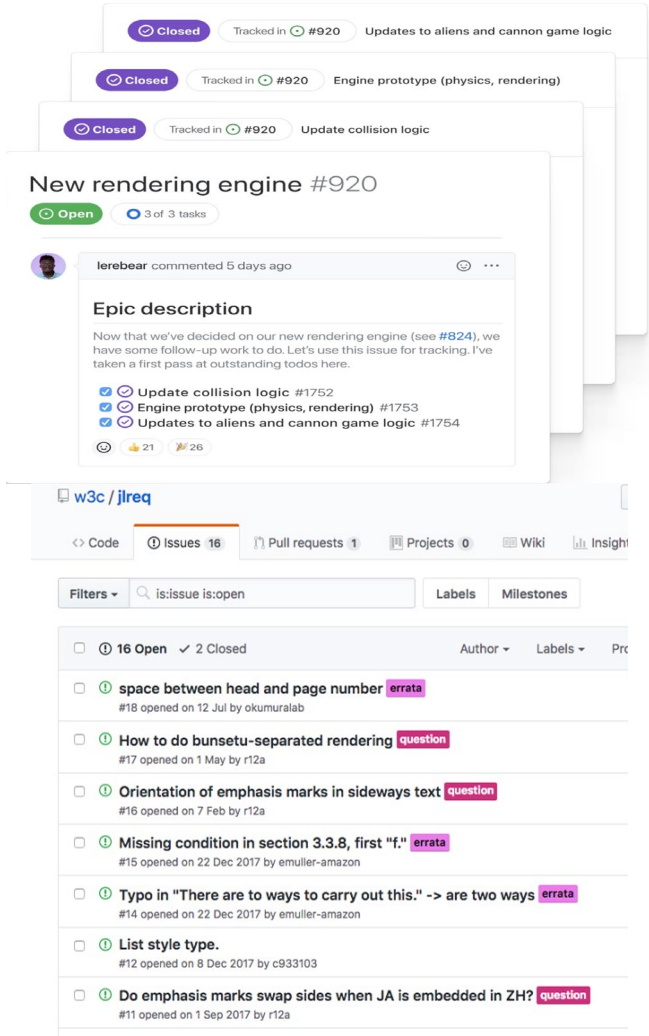


# Issues

- GitHub Issues is a feature of the GitHub platform that enables developers and teams to track bug reports and pull request.

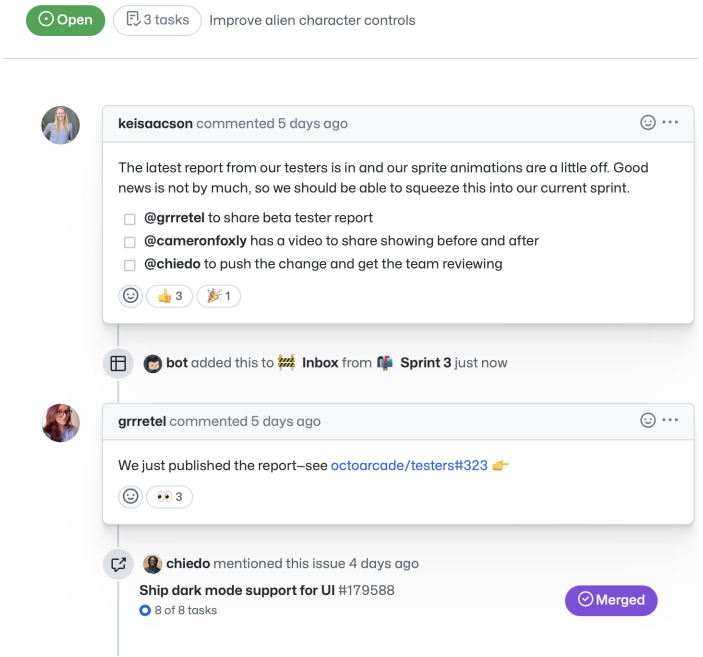
- When a user identifies an issue with a project a ‘ticket’ will be created.

- Other members of the team can then review the Issue, ask questions, or provide suggestions for how to resolve it



# Issues

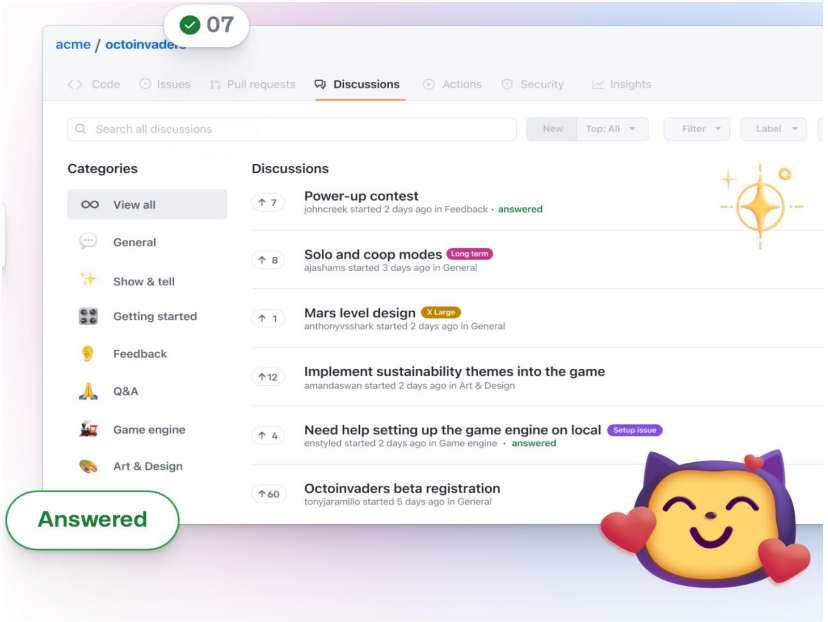
- Issues have threads attached to them that allow a space for discussion for the issue at hand
- By providing a clear record of who said what and when, issues have threads that can help ensure that everyone involved in the discussion is held accountable for their contributions.
- Once the Issue has been resolved, it can be closed, either manually or automatically



# Discussions

- Discussions is a feature of the GitHub platform that allows project maintainers and contributors to have discussions about a project.

- Discussions provide a space for people to talk about things that don't fit neatly into the project's issue tracker or pull request workflow, such as:
  - feature requests
  - project governance
  - community guidelines
  - general questions or feedback.

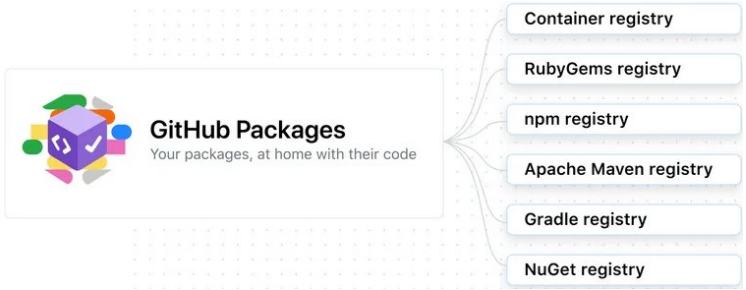


# Packages

- GitHub's package feature is a simple and straightforward way to publish, share, and consume packages of code, libraries, and other software components.

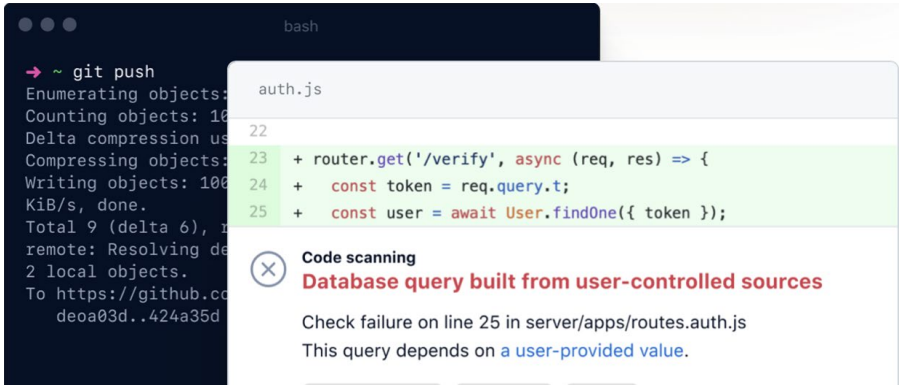
- A Package is a collection of files that are bundled together and can be installed and used by other developers.

- Once your package is published on GitHub, other developers can easily install and use it by adding a reference to the package in their own code or project



# Security

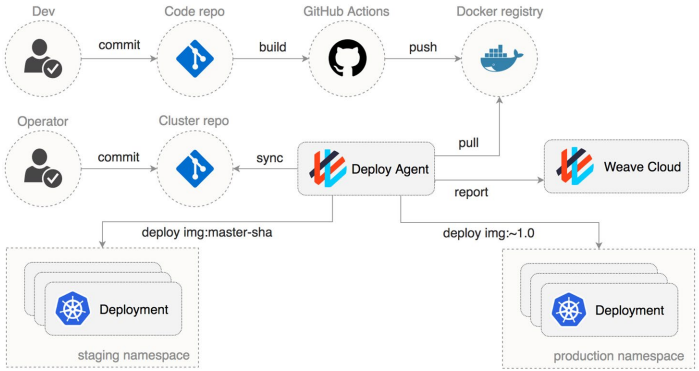
- Repository permissions: GitHub allows repository owners to grant different levels of access to collaborators
  - read-only access, write access, and administrative access.
- Code scanning: a feature that scans repositories for potential security vulnerabilities
- Dependabot: Dependabot is a GitHub feature that automatically scans repositories for outdated or vulnerable dependencies
- Two-factor authentication
- Encrypted data storage

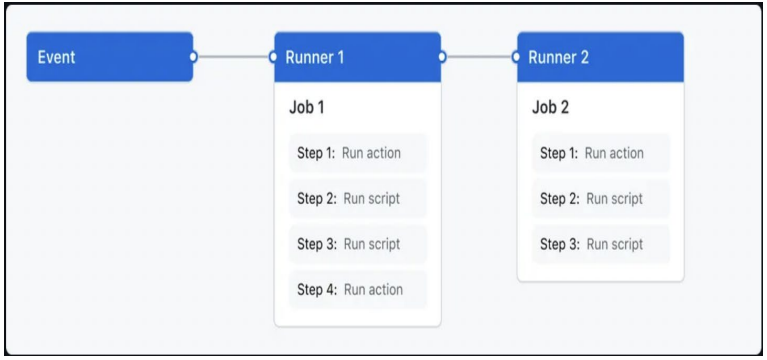


# Actions

- GitHub Actions is a continuous integration and continuous delivery platform that allows you to automate your build, test, and deployment pipeline.

- GitHub Actions lets you run workflows when other events happen in your repository: you can run a workflow to automatically add the appropriate labels whenever someone creates a new issue in your repository.



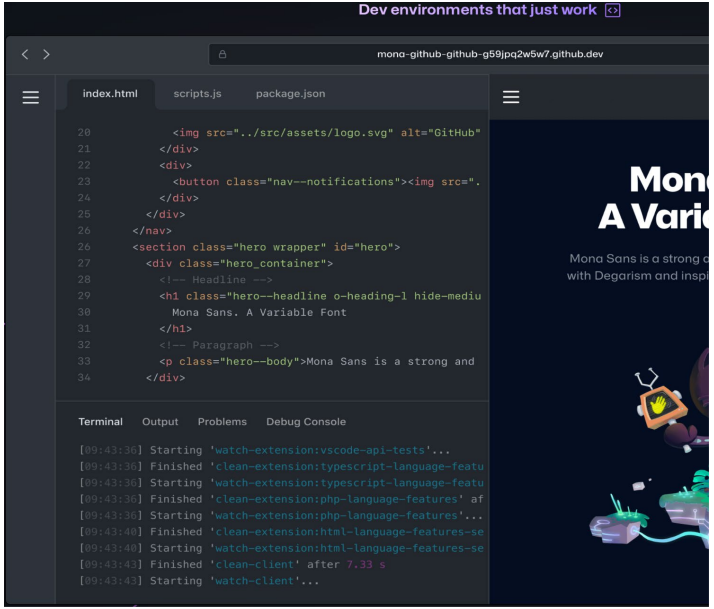



# CodeSpaces

- Codespaces offers a GitHub CodeSpaces, a cloud-based development environment that allows developers to write, run, and debug code directly within their GitHub repositories.
  - Code from any device
  - Model, train, and analyze data
  - Fix bugs right from a pull request

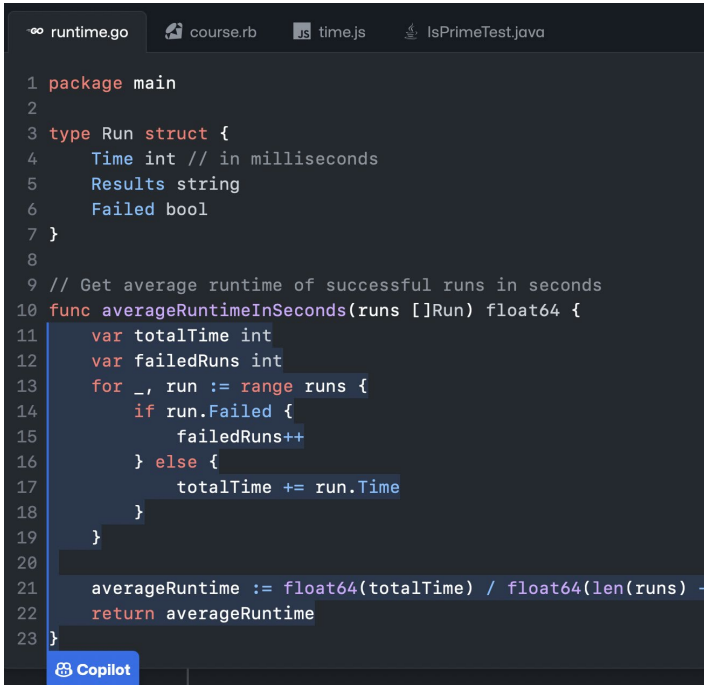
- CodeSpaces are designed to be flexible and customizable, allowing developers to choose from a variety of pre-configured environments or create their own custom environments.

- The key benefit of CodeSpaces is that they allow developers to collaborate on code in real-time, without the need for local installations or complex setup procedures.



# CoPilot

- GitHub Copilot uses the OpenAI Codex to suggest code and entire functions in real-time, right from your editor.
- GitHub Copilot turns natural language prompts into coding suggestions across dozens of languages.
- This can help you learn the nuances of a language you might be learning and deepen your understanding with ai generated comments.

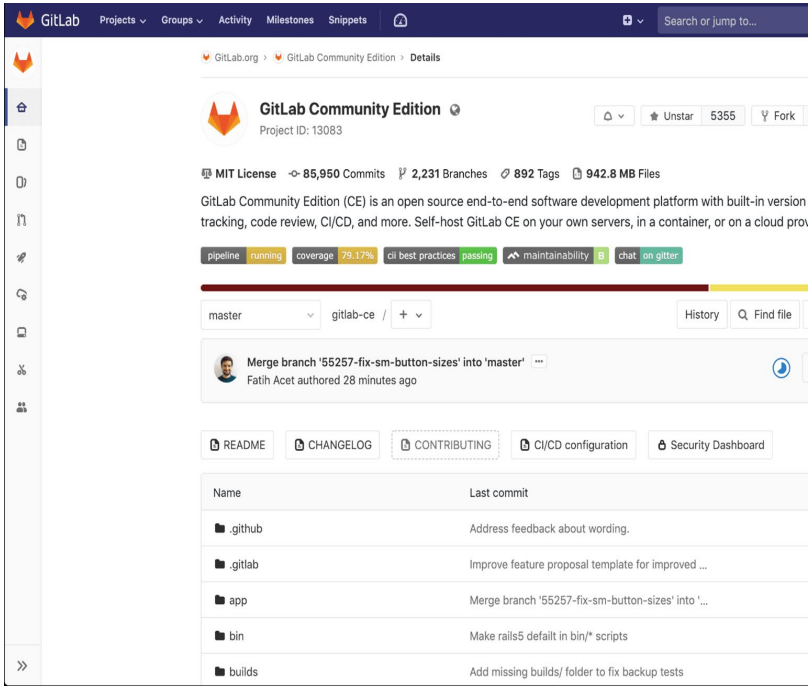


# Why not just use git?

- Git is a powerful tools, but github provides far too many features to be ignored:
- Cloud hosting
- collaboration features
- Community
- CI/CD tools
- Security
- These features make github more than just a version control system

# What is GitLab?

- Git lab is a team based, security focused, production-management system for software development.
- The open project helps millions of developers work together in a collaborative virtual environment.
- Much like GitHub, GitLab is a hosting platform for software development but in a more team oriented way.
- The platform is trusted by various fortune 500 companies.



# What is GitLab?

T-Mobile

Goldman Sachs

CLOUD NATIVE COMPUTING FOUNDATION

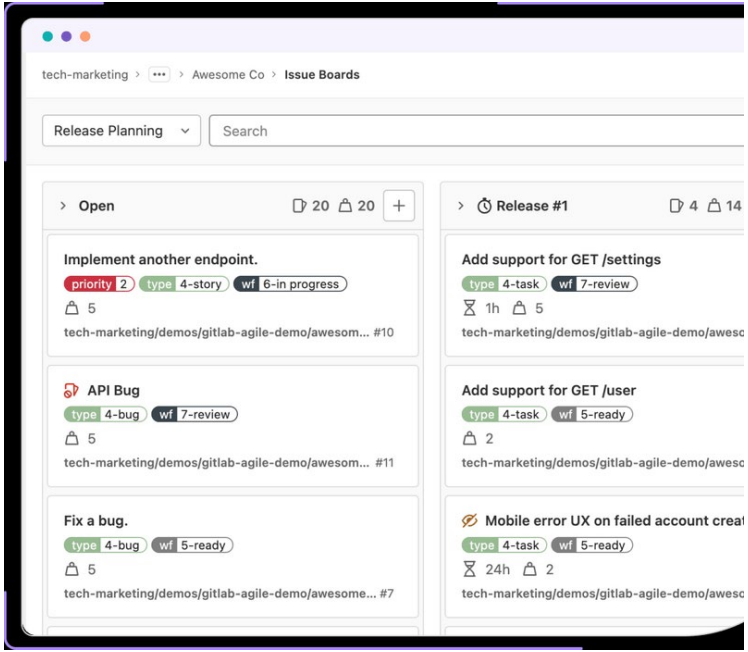
SIEMENS

NVIDIA

UBS

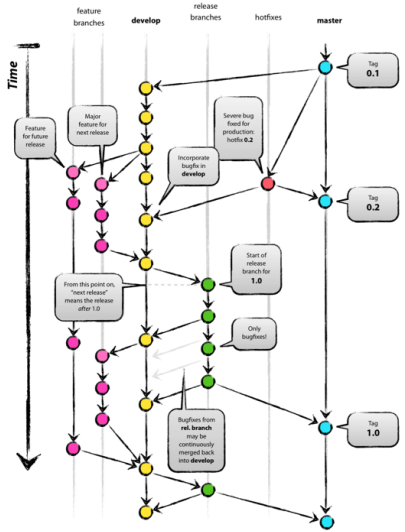
# Planning

- Gitlab features an interface for the team to suggest, plan, and track the implementation of the features and bug fixes of a project.
- Suggestions
- Priority



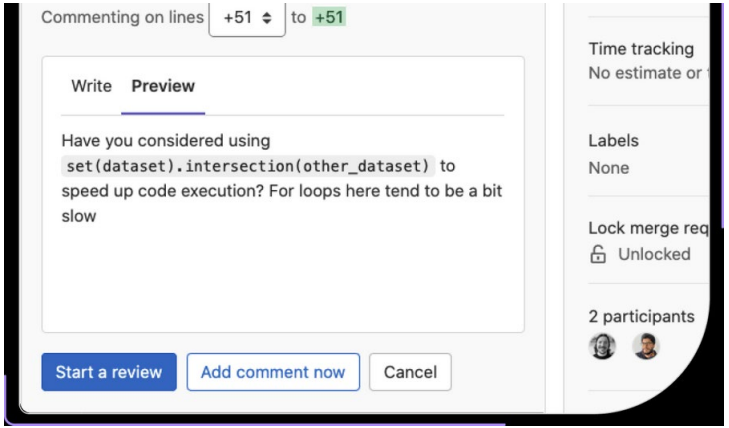
# Planning

- GitLab's version control system helps teams manage changes to their codebase, enabling collaboration and tracking changes over time.




- File locking and source access control

- Support milestones and code reviews

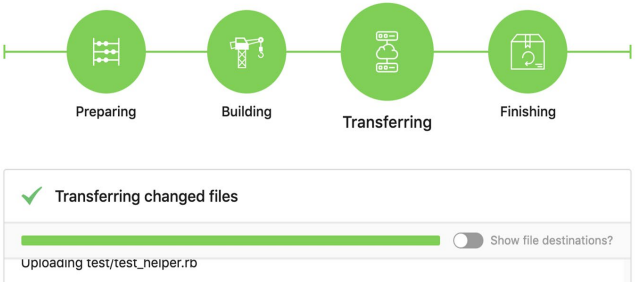


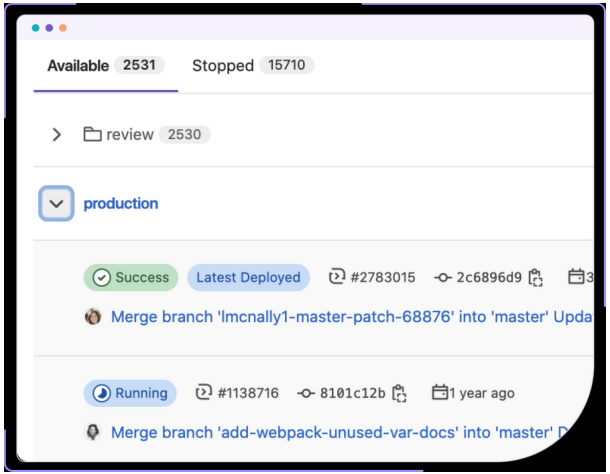
# Automatic Releasing

- Gitlab a helps automate the release and delivery of applications that streamlining manual processes.

- Once the code passes all the tests and meets certain predefined criteria it can be automatically released to the production environment.

- With zero-touch Continuous Delivery built right into the pipeline, deployments can be automated to multiple environments like staging and production, and the system just knows what to do without being told.



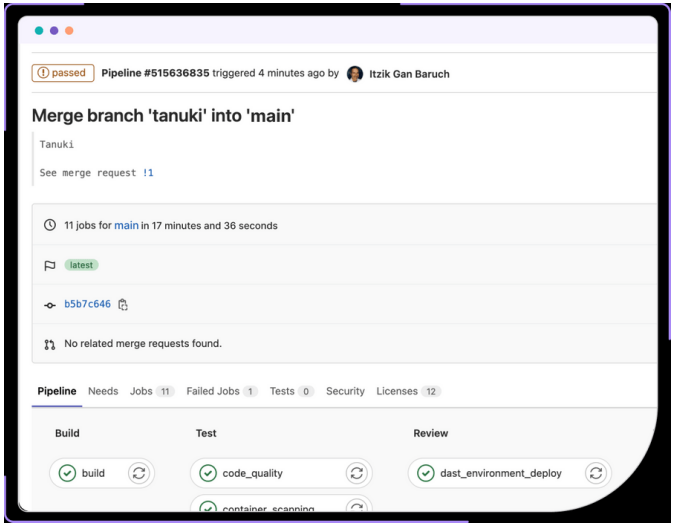



# Code Integrity/quality

- Gitlab supports automated testing for code integrity/quality.

- These tests include:
  - Static Analysis
  - Security Testing
  - Dynamic Analysis
  - Code Quality Analysis

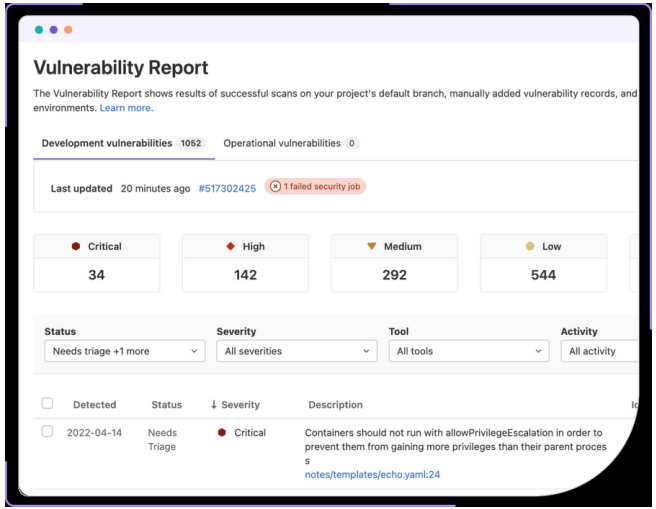
- These tests allow project managers to provide fast feedback to developers and testers about the quality of their code.



# Security

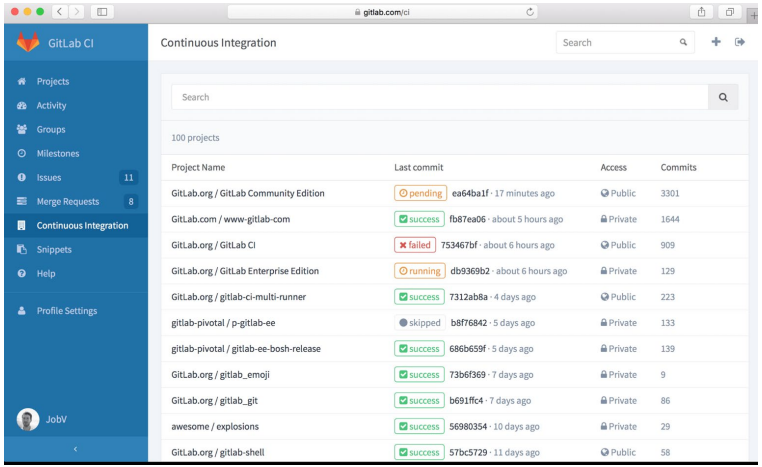
Security features are integrated directly into the development life cycle. Letting project manages utilize the following:

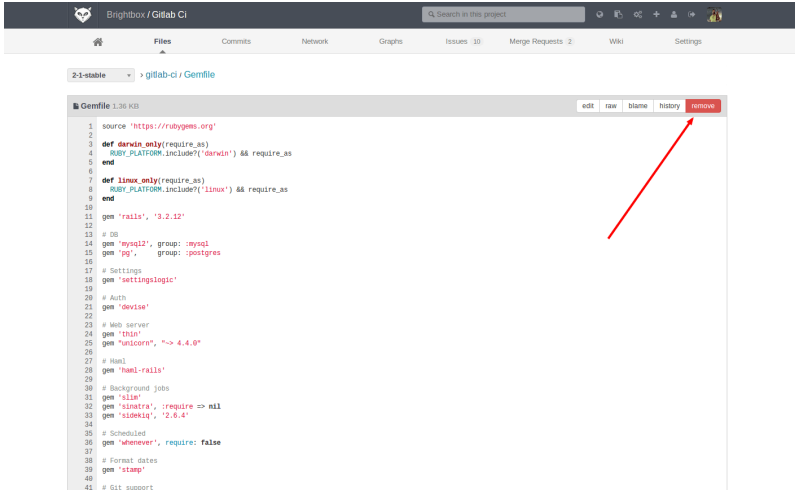
- Static Application Security Testing (SAST)
  - Scans applications code to find the root cause of vulnerabilities from “the inside out”
- Dynamic Application Security Testing (DAST)
  - Simulates attacks throughout the front end to detect vulnerabilities from the ”outside in” testing
- Container Scanning
  - Test for vulnerabilities on possible hosts for the software
- Dependency Scanning
  - Scans application and system dependencies for



# Pros of GitLab

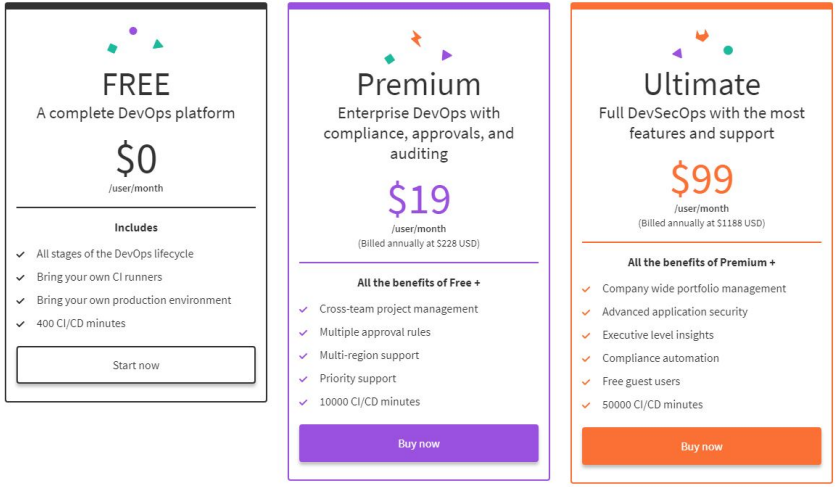
- Comprehensive Streamline toolset
  - Automatic testing
- Open source
- Self or cloud hosted
- Strong security features
  - Role based project management
- Large active community of dev's
  - documentation & online resources
- Continuous development and deployment



# Cons of GitLab

- Steep learning curve
- Resource-intensive
- Limited free version features
  - The free version comes with limitations on the number of users and projects, as well as some feature restrictions.
- Limited customization
- Lack of third-party integrations
  - Certain projects that depend on  may not work with the framework



# DEMO