2020

A BEGINNER'S GUIDE

Source code management for the enterprise

Fifteen best practices to help large software teams innovate and collaborate

- ✓ Enterprise development challenges
- ✓ The impact of source code management
- ✓ Best practices
- ✓ GitLab for enterprise source code management

GitLab

Table of contents

Introduction 01

Enterprise development challenges 03
- Large teams 03
- Distributed teams 03
- Massive projects 04

The impact of source code management 05
- Collaboration 05
- Security 05
- Visibility 05
- Quality 06
- Delivery and velocity 07

SCM Best practices 07
- Use one repository per application or library 07
- Manage access control with groups 07
- Document everything in the issue tracker 08
- Use SCM project planning tools 08
- Keep the backlog small 09
- Keep issues small 09
- Create sprints 09
- Use feature branches 10
- Make small, frequent commits 10
- Make code review mandatory 10
- Embrace DevOps by using CI/CD 11
- Build robust, automated testing into CI/CD pipelines 11
- Build security into the CI/CD pipeline 12
- Use SCM reporting to communicate progress 12
- Use industry-specific SCM extensions 12

GitLab for enterprise source code management 13
- Comprehensive planning 13
- Built-in CI/CD 10 14
- Kubernetes integration 14
- Detailed reporting 14

Conclusion 15

Introduction

Software teams at large enterprises face increasing challenges, because they can't infinitely scale old development processes.

Growth inevitably causes software development problems, which enterprises may have difficulty solving when balancing the pace of innovation with the time and effort required to create new processes or workflows.

Fortunately, there is a simple solution in using source code management. Nearly every enterprise uses source code management (SCM) in some way.

The main draw of SCM in the enterprise is that all of the company's code is stored and managed in a controlled way.

A complete history of code commits, along with release and version tracking, lets managers see the status of a project and understand how it has evolved over time. Beyond that, most enterprises view source code as valuable intellectual property. A good source control system ensures intellectual property is tracked, stored, and backed up.

Source code management goes beyond just code to include feature and issue tracking. These help all stakeholders participate in the development process.

- **Product owners** can see which feature requests have been accepted, and view the order in which the features will be completed.
- **Product managers** can track the progress of individual tasks associated with the delivery of new features. They can use this information to ensure the overall project is on schedule, and assign extra resources if any parts of the project are falling behind.
- **Developers** can see the status of the project, and see the tasks assigned to them so they know what to work on next. This is particularly important for distributed teams, because team members can't always communicate in real time.
- **Development managers and team leads** can assign tasks to every developer based on the developer's skill set and availability.
- **End users** can adopt the issue tracker to report bugs and other problems they've encountered.

This eBook focuses on the impact of source code management and best practices to help large software development teams accelerate delivery, streamline collaboration, and maintain a clean codebase.

Enterprise development challenges

Some of the biggest challenges enterprises face include streamlining a workflow with large teams, ensuring collaboration with a distributed team, and managing massive projects. Each of these challenges can have a significant impact on how successful an enterprise development team meets business objectives and customer demand.

Large Teams

Large enterprises face difficult business challenges requiring complex solutions, which in turn require sizable software development teams. Enterprise software projects routinely take years to complete, and the resulting applications run for decades. Due to their sheer size, these projects require dozens, hundreds, or even thousands of developers. Organizational culture also inflates the development team size because many enterprises have a tradition of large, hierarchical departments.

Managers commonly have 50 or more direct reports, and in teams this size, it's difficult to ensure all developers work on the highest priority tasks, because a lone manager can't easily keep track of what every developer is working on.

Prioritizing development work requires visibility into exactly which tasks are complete, which tasks are in progress, and which tasks are blocked pending the completion of other tasks. The larger a team, the harder it is to achieve this kind of visibility.

Distributed teams

Large companies often have offices and remote employees distributed around the world to maintain a presence in various markets. In a global organization, it can be difficult to keep everyone in sync if proper communication and collaboration processes are not established.

Large enterprises may not have the time or team members to create strong routines and guidelines to maintain collaboration across time zones, develop communication preferences, and implement ways to provide end-to-end visibility.

Without strong communication and collaboration, information silos develop, which hinder the software development lifecycle and lead to delayed releases.

Massive projects

Enterprise software projects commonly have hundreds of contributors, and delivery dates are often years in the future. It's tough to ensure projects this size are delivered on time and budget, and perform as expected.

With such a distant delivery date, it can be difficult to get teams to work quickly and efficiently, because developers may feel like there's plenty of time for everything.

Complacency at the beginning of a project leads to stress and 80-hour work weeks at the end of the project. Large projects usually come with large budgets, which invite large expectations. Teams may have to rush to meet goals rather than spend time innovating and iterating on exciting solutions.

The impact of source code management

Source code management has the power to improve enterprise software development. Here's a look at how SCM impacts various aspects of large enterprise organizations.

Collaboration

Enterprise development teams are often large and geographically distributed, making collaborative coding essential.

Developers can use SCM branches to develop features and fix issues in isolation from a project's main branch, helping each developer focus on the task at hand, without worrying that they'll interfere with the work of other developers. Source code management also helps geographically distributed teams to work on code effectively. A distributed team member can't always ask colleagues about the status of a project, because their colleagues and managers on the other side of the world may be asleep.

- Branches and merge requests also make code review easy. Code reviews ensure that multiple people see each line of code before it is merged back to the main branch. This practice ensures that all code meets the company's style guide and includes a full set of unit tests.

Security

Most large enterprises want to maintain strict control over who can see its source code. A good SCM provides access control to ensure only people authorized to see an application's code can pull it from the repository.

Access control can be set at the repository level, so a developer can be granted access to only those repositories they need to work on.

Beyond controlling who can see the code, SCM systems also make it easy to limit who can commit code to a repository's main branch, which is especially important when CI/CD systems are set up to automatically build and deploy commits to the main branch.

Visibility

Enterprise software projects typically have milestones and deadlines, and managers are accountable for meeting these expectations. A fully-featured SCM provides project management tools to follow a project's progress and track completion of individual sprints and milestones.

Quality

A good SCM helps improve code quality by making it impossible to sneak unreviewed code into the repository. The best way to ensure no untested, low-quality code makes it into the build is by making it impossible to commit directly to the main branch.

All new code must be added in branches, and merged into the main branch via a merge request after the code has been reviewed and approved.

Delivery and velocity

The first step in delivering great software quickly is access control. SCM systems ensure that only authorized developers can work on code, and no one can add poor, untested code directly to a repository. Developers spend more time moving the codebase forward, and less time fixing problems that shouldn't have made it to production.

But code sitting in a repository isn't helpful by itself. The code must still be built, tested, and deployed. For this reason, most SCM tools can be easily connected to a continuous integration/continuous delivery (CI/CD) system.

The CI system pulls the latest code from the SCM repository, builds it, and runs all of its tests. If the code builds and the tests pass, the CD system automatically deploys the build to production. This approach lets enterprises deploy dozens of times a day. That's a significant improvement over the standard enterprise practice of dozens of days per deployment.

SCM best practices

There are several best practices that will help teams make the most of SCM in the enterprise. Each of these fifteen best practices can help solve the most common challenges that enterprises experience.

1 Use one repository per application or library

While it's easy to keep all the company's code in a single repository, large enterprise development teams usually work on multiple projects at once.

Consider creating a separate repository for each application or library the team works on, so that it's easier to control access to each repository, and prevents developers from making changes to code someone else is responsible for.

It also makes code review and CI/CD processes more effective, as changes to each repository can be monitored separately.

2 Manage access control with groups

Modern SCM tools offer the ability to group multiple code repositories together.

Grouping related repositories offers easy access control as permissions can be granted at the group level.

All developers granted access to a group can view the corresponding repositories. In large enterprise development teams, not every developer needs to see all of the company's code. If repositories are organized into logical groups, it only takes a single click to grant or revoke a developer's access to multiple repositories.

3 Document everything in the issue tracker

The best software teams are those that understand the business problems they're solving.

Documenting everything in an SCM issue tracker and grouping related issues together keeps all team members informed.

"Everything" means requirements, technical specifications, bugs, support calls, and user documentation. Recording this information where everyone can see it ensures that all team members can easily understand their project's status, regardless of their physical location.

4 Use SCM project planning tools

Large projects are best developed as a series of smaller sub-projects. To keep these projects organized, enterprise source code management typically offers integrated project management tools. Most SCMs have a way to record high-level business goals as epics for each project.

If a project is complex, use child epics to break large goals into manageable pieces.

Using business-friendly epic names helps non-technical stakeholders understand a project's components. For example, a large epic called "HR Management System" might have child epics like "Payroll Management Application" and "Recruiting Application." Consider grouping deeply technical epics under a parent epic such as "Common Utilities" or "Technical Debt."

Under each epic or child epic, use the SCM issue tracker to record the user stories and requirements needed to achieve each business outcome. When projects are underway, use SCM features like milestones to track both individual sprints and overall progress toward project completion.

5 Keep the backLog small

The backlog is typically a “catch-all” epic that ensures important tasks not yet assigned to an epic are not overlooked.

As projects progress, backlogs tend to increase. Large backlogs often have hidden epics within them, so it’s important that teams review backlogs periodically to identify hidden tasks.

Ideally, every issue that represents a project task is assigned to an epic, which may sound like common sense, but misplaced issues are a reality, and they usually end up in the backlog.

6 Keep issues small

Every task should be broken down into small, manageable changes. The larger a task, the harder it is to estimate how long it will take to complete and the easier it is to miss important details. Developers touch more code for each task, increasing the opportunity for creating new bugs.

Small tasks enable more accurate estimates of how much work is required to complete the task successfully.

Small tasks also help ensure developers are solving the right problems by focusing on completing small, descriptive user stories. Working against a backlog of small, clear tasks helps teams increase development velocity.

7 Create sprints

Sprints are collections of tasks to be completed in a fixed time-frame—usually one to three weeks, although sprint length isn’t set in stone. Most development teams find two-week sprints ideal, so that’s a good default to start with.

Sprints help teams pace themselves and work with urgency—not stress—throughout the project, even when the delivery date is six months or more in the future.

Completing all of a sprint’s tasks in the allotted time should be celebrated. And if the tasks aren’t completed, it’s not a failure - it’s a learning opportunity. The team can evaluate the size of the remaining tasks, and use that information to better estimate future sprints.

Keeping feature branches small and short-lived ensures they can’t grow large and contain too many commits.

Feature branching prevents review fatigue, which happens when code reviewers let mistakes slip by because they’re overwhelmed by the volume of new code.

8 Use feature branches

When adding new features, consider using a short-lived `feature` branch for each feature. Each branch should live no longer than a sprint. Avoid creating one branch for an entire sprint that is used by all engineers assigned to the project.

Keeping feature branches small and short-lived ensures they can't grow large and contain too many commits.

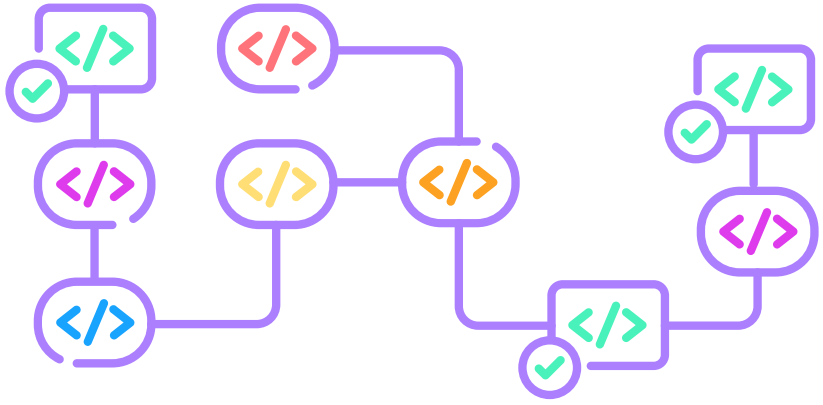
Feature branching prevents review fatigue, which happens when code reviewers let mistakes slip by because they're overwhelmed by the volume of new code.

9 Make small, frequent commits

Encourage developers to make small, frequent commits when adding features or fixing bugs.

Iterative development helps teams understand functionality, receive fast feedback from users, and identify future areas of improvement.

Small commits make it easy to roll back changes if problems occur and ease code review, making it easier for reviewers to understand why each piece of code was added.

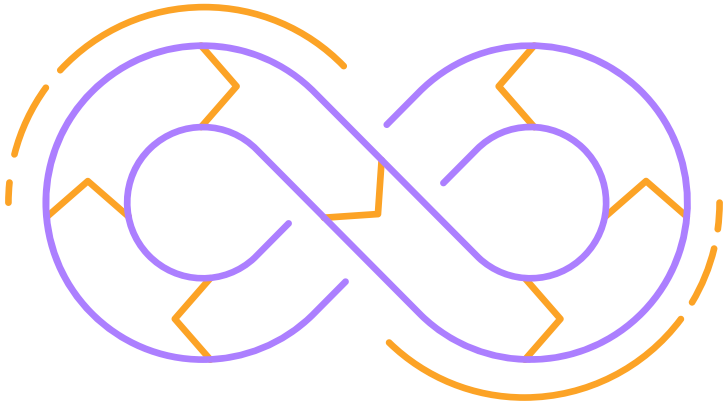


10 Make code review mandatory

Code review should be mandatory before merging any code into a repository's main branch.

Code reviews prevent poorly written, untested code from entering into a codebase.

Ensure that all code is reviewed by at least one other developer, no matter who wrote it. Even the most senior developers make mistakes.



11 Embrace DevOps by using CI/CD

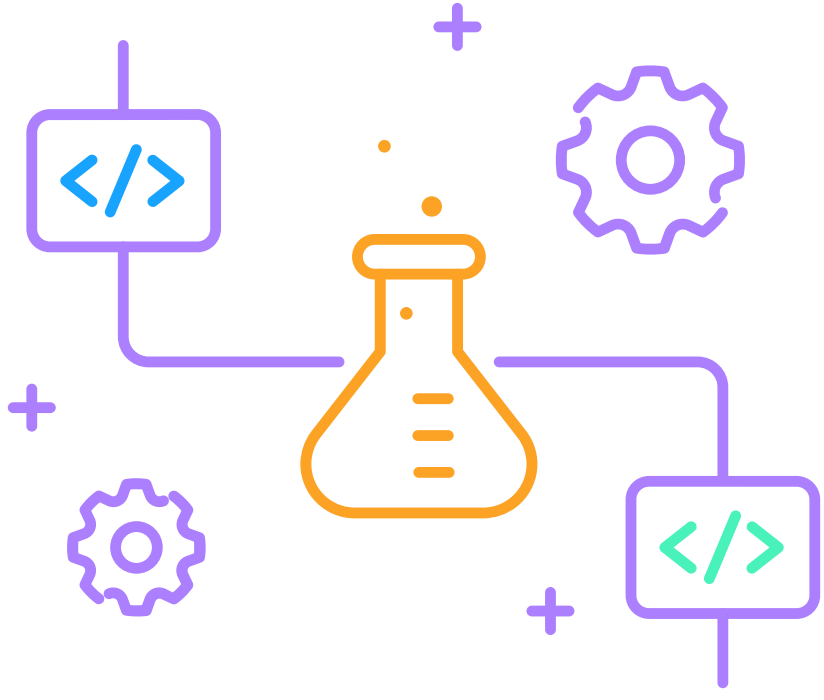
Development and operations teams should not exist as silos.

The most effective enterprise SCM tools offer CI/CD integration or have CI/CD built-in to enable effective DevOps practices by ensuring a smooth, automated flow that quickly turns raw source code into deployed applications.

12 Build robust, automated testing into CI/CD pipelines

Automated testing—in the form of unit and integrations tests—ensures that applications perform as expected.

Set up the CI/CD pipelines to run tests after every push to a branch and after every merge request. Consider automatically failing merge requests if the code changes in the request cause a test failure so that reviewers know right away that they shouldn't spend time reviewing the code until the test failure is resolved.



13 Build security into the CI/CD pipeline

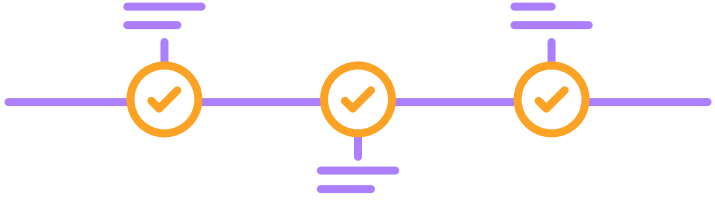
Many CI/CD integrations are capable of using third-party tools to check for security vulnerabilities.

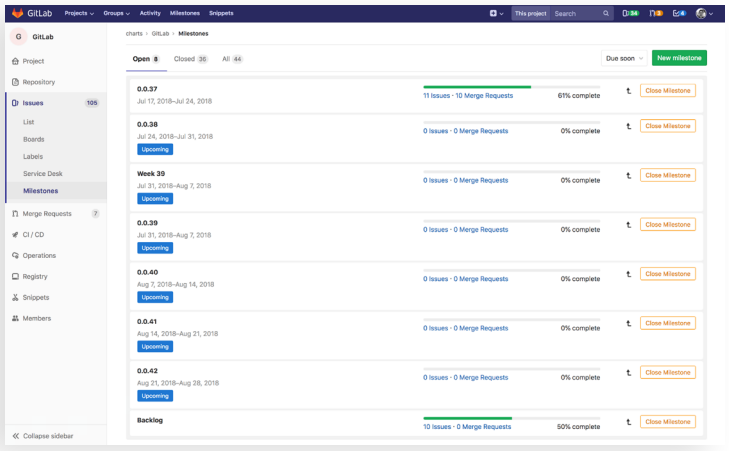
Consider adding tools to the CI/CD pipeline to prevent security issues from making it to production. Shifting security left is an important aspect in maintaining a secure codebase.

14 Use SCM reporting to communicate progress

Enterprise software projects are often opaque, offering no insight into how the project is progressing toward solving business goals, which frustrates stakeholders counting on the project's results.

Modern SCM systems contain reporting tools that provide the insight stakeholders want to see. Burndown charts are a common technique to determine when a sprint (or milestone) will be complete and are a useful way to illustrate progress.





Milestones are a way to track issues and merge requests created to achieve a broader goal in a certain period of time.

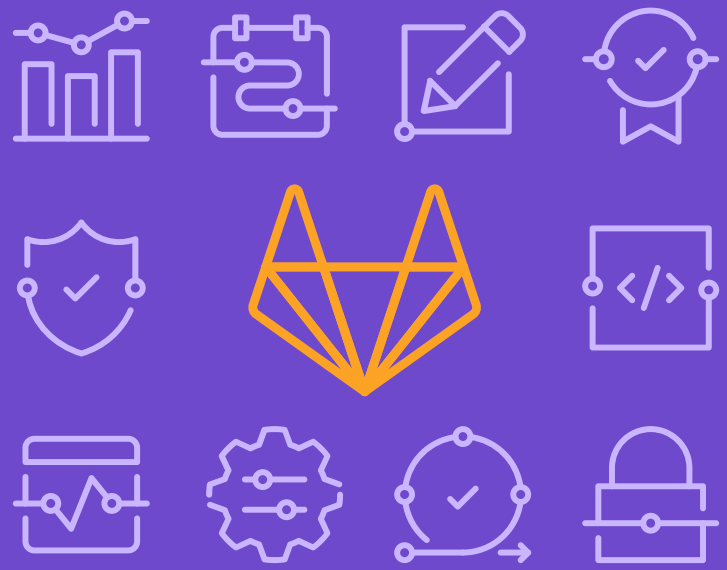
15 Use industry-specific SCM extensions

Most SCM systems are loaded with functionality by default, but they don't always cover every use case. For example, many software projects in fields like GIS, AI, and gaming need to track and version large binary files alongside source code. Although most SCM systems don't support this natively, extensions like Git Large File Storage enable teams to manage important binary assets in the source repository instead of storing and tracking them separately.

GitLab for enterprise source code management

GitLab's features can help enterprise teams deliver better software faster.

Whether the enterprise team is distributed or colocated, GitLab's complete set of Agile planning tools makes it easy to coordinate developers and complete projects on time.



Start your GitLab free trial

Comprehensive planning

Development starts with GitLab Issues, which represent a single feature that will deliver business value to the enterprise. Team members can collaborate via messages in issues if they have questions or need more information. Issues that share a theme are grouped into epics, which can be combined into roadmaps that make it easy to visualize everything the enterprise's developers are working on.

Put together, GitLab Issues, Epics, and Roadmaps ensure that all stakeholders—from developers to executives—understand what's happening at a high level, and what's planned for the future.

At a tactical level, GitLab Milestones help developers understand exactly what they should be working on right now. Milestones usually represent an Agile sprint, and contain a list of issues the team expects to complete during the sprint. Milestones are great for managers, too. At a glance, a manager can tell how much progress has been made toward a milestone and see which issues and merge requests are still outstanding.

Built-in CI/CD

GitLab includes an integrated CI/CD system, offering an all-in-one enterprise software delivery platform.

In addition to hosting code and managing a product, GitLab can also build, test, and deploy applications to production.

Kubernetes integration

Enterprises increasingly use software containerization tools like Kubernetes, so DevOps teams need a place to securely store container images.

Fortunately, GitLab offers a container registry, making it easy for development teams to develop and test using the same container environments the operations teams deploy to production.

GitLab also has complete Kubernetes integration, so teams can deploy straight from GitLab to an enterprise Kubernetes cluster. There's no need to cobble together a pipeline of mismatched applications to get code built and running—GitLab does it all. With built-in Kubernetes monitoring, teams can track the performance of production applications to ensure they're behaving as expected, providing visibility long after development.

Detailed reporting

Reporting is particularly important in the enterprise, as most software projects have managers and executives depending on the project's success.

GitLab offers a variety of tools to efficiently and easily gain insight into projects.

At the project level, tools like burndown charts show progress toward competition of sprints and other milestones. Productivity Analytics provides deep insight into the productivity of individual developers, and Contribution Analytics provides similar insight at the team level. Finally, Value Stream Analytics helps leaders understand how long it takes development teams to deliver business value—from initial idea to final deployment.

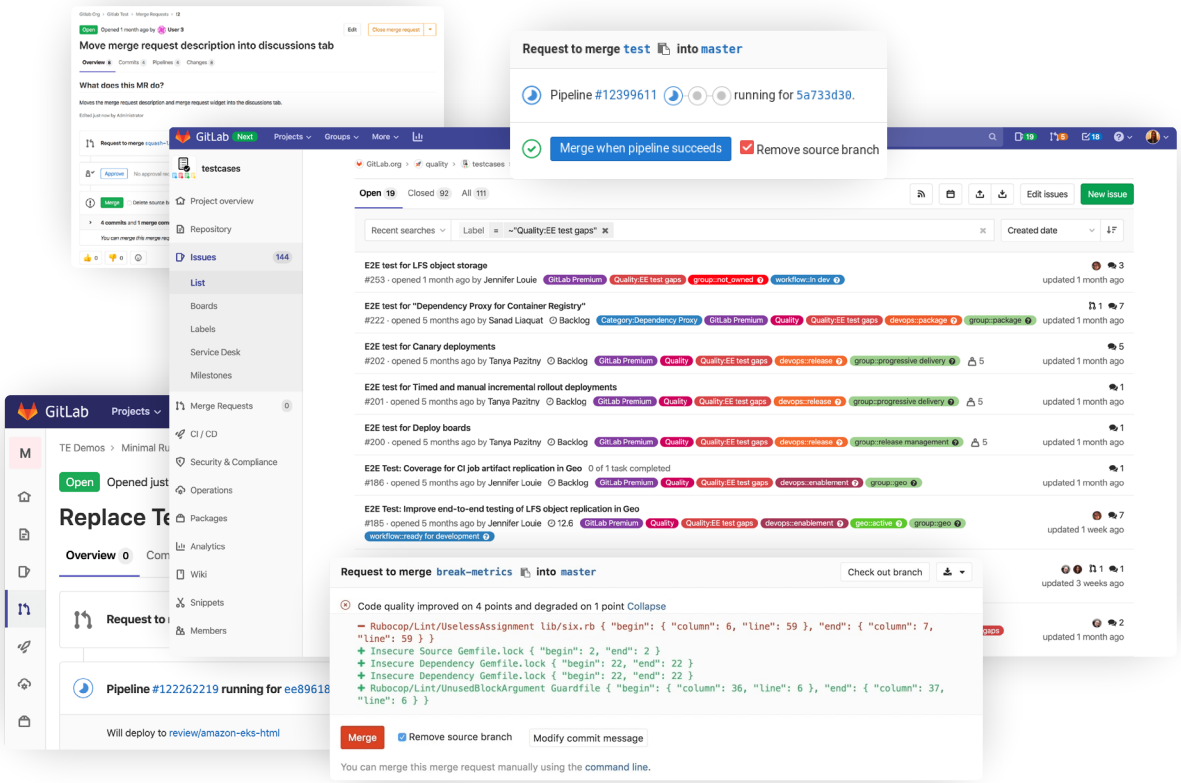
Conclusion

Deliver more software with GitLab

GitLab is a fully integrated environment providing end-to-end management of software delivery, helping teams coordinate remote staff, communicate with the business, maintain code quality, manage dependencies, and ensure security and compliance.

GitLab helps enterprises deliver software more quickly, more effectively, at lower cost.

With GitLab, everything is in one tool and teams of all sizes can move fast and deliver value as quickly as a nimble startup. can propel development teams to faster delivery, stronger collaboration, and higher velocity.



Try GitLab free

All GitLab features — free for 30 days.
GitLab is more than just source code management. It is a full software development lifecycle & DevOps tool in a single application.

Start your free trial

About GitLab

GitLab is a DevOps platform built from the ground up as a single application for all stages of the DevOps lifecycle enabling Product, Development, QA, Security, and Operations teams to work concurrently on the same project.

GitLab provides a single data store, one user interface, and one permission model across the DevOps lifecycle. This allows teams to significantly reduce cycle time through more efficient collaboration and enhanced focus.

Built on Open Source, GitLab leverages the community contributions of thousands of developers and millions of users to continuously deliver new DevOps innovations.

More than 100,000 organizations from startups to global enterprises, including Ticketmaster, Jaguar Land Rover, NASDAQ, Dish Network, and Comcast trust GitLab to deliver great software faster.

GitLab is the world's largest all-remote company, with more than 1,250 team members in more than 65 countries and regions.

Learn more at GitLab.com