2020

Version control best practices

Five easy ways to enhance team collaboration

- ✓ Determine a branching strategy
- ✓ Make frequent small changes
- ✓ Write descriptive commit messages
- ✓ Develop using branches
- ✓ Conduct regular code reviews

GitLab

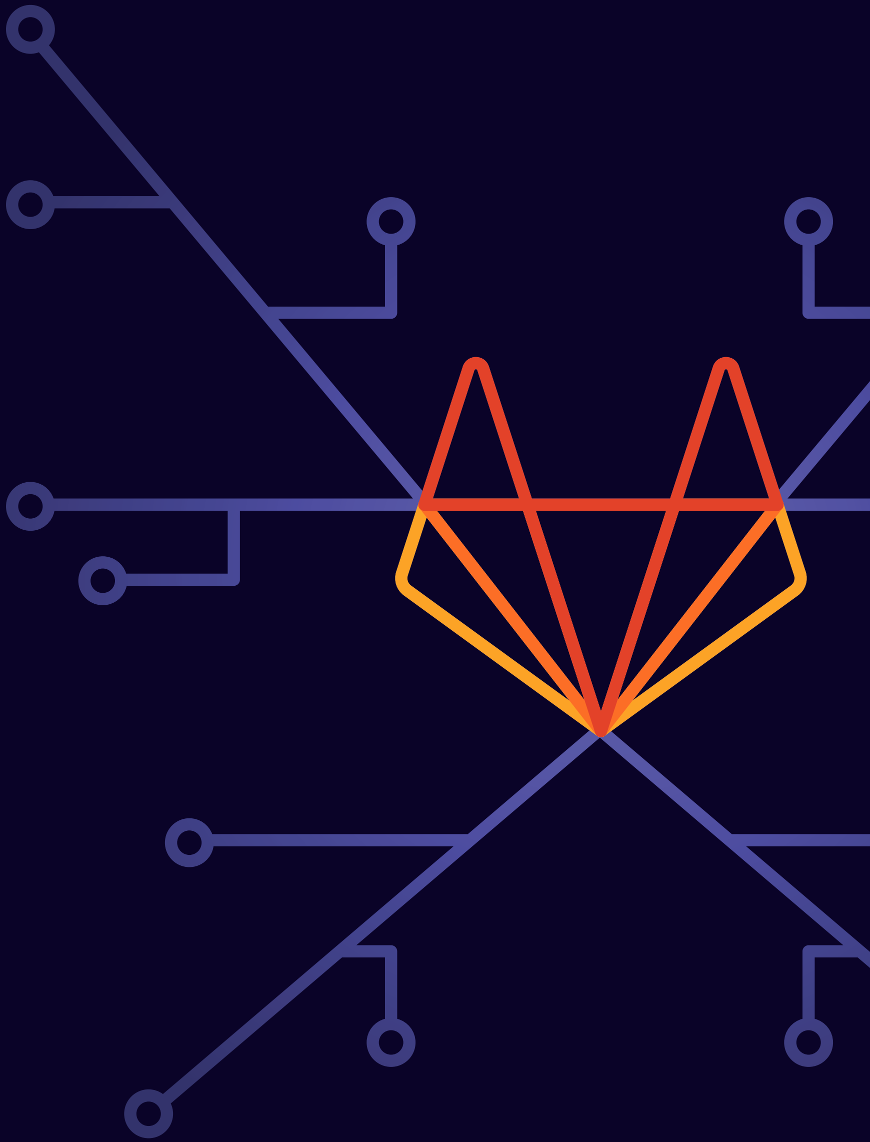


Table of contents

Introduction 01

Part One: Determine a branching strategy 02

Overview
Impact on collaboration

Part Two: Make frequent, small changes 04

Overview
Impact on collaboration

Part Three: Write descriptive commit messages 05

Overview
Impact on collaboration

Part Four: Develop using branches 06

Overview
Impact on collaboration

Part Five: Conduct regular code reviews 07

Overview
Impact on collaboration

Version control and collaboration with GitLab 09

Start your GitLab free trial

Introduction

Rapid changes in industry combined with increasing customer demand for new features can lead teams to work in silos. Application development requires speed and iteration, making seamless collaboration a necessity in delivering business value. Teams turn to version control to streamline collaboration and break down information silos.

Version control coordinates all changes in a software project, effectively tracking changes to source files, designs, digital assets, and metadata to help teams quickly collaborate and share feedback, leading to immediate, actionable changes.

Teams that embrace version control and collaboration can seamlessly coordinate work, review changes, and manage delivery, helping them focus on solving problems and shipping value.

Version control and collaboration goes beyond simply tracking changes and includes practices such as:

- Enabling development teams to work in distributed and asynchronous environments
- Managing changes and versions of code and artifacts
- Enabling code reviews and other assets
- Tracking approvals of proposed changes
- Resolving merge conflicts and related integration issues

Getting started with version control can be a challenging task, especially when faced with a rapidly changing environment. This eBook presents five best practices to help development teams strengthen collaboration to iterate on new features and deliver business value using Git.

Part One

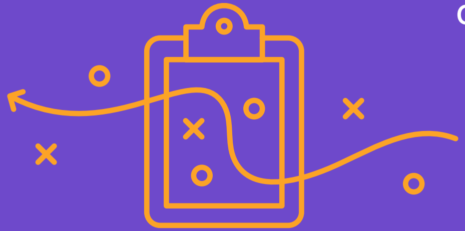
Determine a branching strategy

Overview

When team members from diverse professional and educational backgrounds work together, there can be conflicting approaches to workflow. To prevent a chaotic development experience, leaders should identify and widely communicate a single branching strategy.

Determining a branching workflow depends on several factors, including team size, level of experience, scaling requirements, and industry limitations.

While teams may choose to follow an established workflow, it's also possible to create a customized workflow based on specific needs. Whichever strategy selected, it's important to communicate the workflow to the team and provide training if necessary.



Common workflows include:

Centralized workflow

A centralized workflow includes a single repository and a master branch. Teams don't use any other branches for development, so there is a high risk of overwriting changes.

Feature branching

Feature branching means creating a new branch for each feature that needs to be added. Everyone who needs to work on the feature commits to the `feature` branch.

GitFlow

GitFlow is an extreme version of feature branching. Developing with GitFlow incorporates a `master` branch and a separate `develop` branch, as well as supporting branches for features, releases, and hotfixes. Development occurs on the `develop` branch, moves to a `release` branch, and merges into the `master` branch.

Task-branch development

GitLab Flow is an example of this type of development and combines feature-driven development and `feature` branches with issue tracking. GitLab Flow provisions multiple environments like staging, pre-production, and production by using separate dedicated branches so that commits flow downstream to ensure that everything has been tested on all environments.

Impact on collaboration

When everyone works harmoniously in the same workflow, there is less risk of overwriting code or breaking `master`.

Furthermore, team members can easily contribute to each other's work, because everyone is familiar with the development and deployment processes. A clear, concise branching strategy sets a rhythm to merge new code and advance projects. This sense of cadence helps team members arrange meetings and manage deadlines and workloads.

Here's a closer look at how each of the popular workflows impacts collaboration:

Centralized workflow

Although there is no collaboration cadence in this type of workflow, it can work well on small teams (fewer than 5 developers) that use good communication to ensure two developers never try to work on the same code simultaneously.

Feature branching and GitFlow

Feature branching connects various teams, because it requires a bigger set of code reviews, push rules, code approvers, and a wider set of tests.

Task-branch development

Task-branch development sets a pretty fast pace, forcing team members to decompose requirements into small chunks of value that will be delivered via `task` branches. This type of workflow embeds collaboration practices such as code snippets, code reviews, and unit testing. Furthermore, if a test breaks, team members can work together to see what's wrong.

Part Two

Make frequent, small changes

Overview

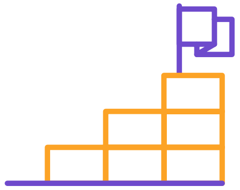
Teams can fall into a habit of committing only when there's a large, near-finished project available, believing that perfection is a prerequisite to delivery.

However, by not taking smaller steps and shipping simpler features, teams risk spending time working on the wrong feature or going in the wrong direction.

The most effective way to develop software with the intention of delivering business value and meeting customer demand is to commit each time you have a working set of tests and code.

Look for ways to simplify projects into small steps and then make frequent commits to complete a larger goal.

Start your GitLab free trial

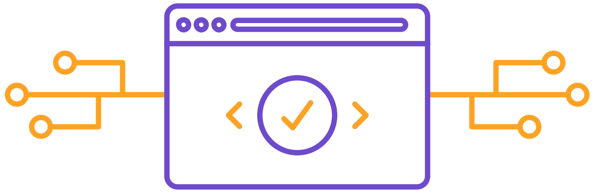


Impact on collaboration

A culture of frequent commits ensures that everyone is aware of what teammates are working on, because everyone has visibility into a code repository.

Teams should frequently push to the feature branch, even when it's not yet ready for review, because sharing work in a feature branch or a merge request can prevent teammates from duplicating work. Sharing work before it's complete also facilitates discussion and feedback, which can help improve the code before it gets to review

Breaking work into individual commits offers context for developers and other teams (e.g. Quality and Security) that will review code at a later time. Smaller commits clearly identify how a feature was developed, making it easy to roll back to a specific point in time or to revert one code change without reverting several unrelated changes.



Part Three

Write descriptive commit messages

Overview

A commit message should reflect intention, not just the contents of the commit. It's easy to see the changes in a commit, so the commit message should explain why those changes were made.

It's important to establish a commit message convention to ensure consistency across teams and decrease confusion and miscommunication.

An example of a good commit message is:
"Combine templates to reduce duplicate code in the user views."

The words *"change," "improve," "fix,"* and *"refactor"* don't add much information to a commit message.

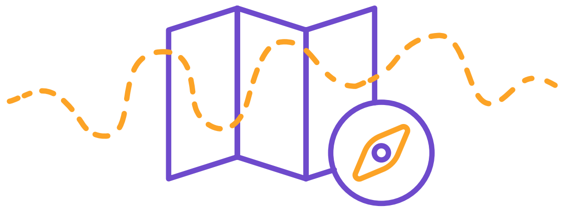
For example, *"Improve XML generation"* could be better written as *"Properly escape special characters in XML generation."*

Impact on collaboration

Descriptive commit messages cultivate transparency and offer insight into progress so that team members, customers, and future contributors can understand the development process.

When conducting code reviews, commit messages help team members follow iterations and determine what changes have been made since a release, discussion, or change in requirements.

Detailed commit messages also assist Quality and Security teams when examining code to identify areas of concern and revert a specific change. Furthermore, when developers write detailed commit messages, it can prevent teammates from duplicating work, limiting delays and helping a project progress more steadily.



Part Four

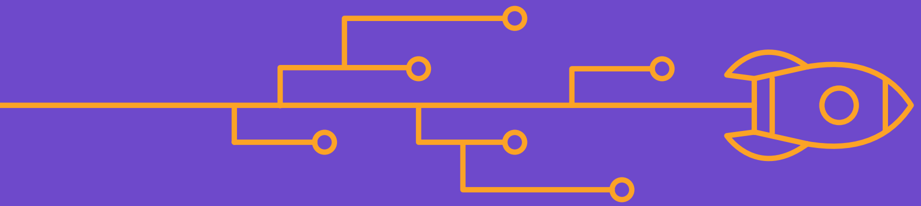
Develop using branches

Overview

Developing in branches is like creating snapshots of a certain branch, usually the `master` branch, at its current state.

Using branches, team members can make changes without affecting the main codebase. The history of changes will be tracked in a branch. When code is ready, it can be merged into the `master` branch.

Coding in branches enables a more organized approach to development and keeps work in progress as a separate draft away from stable, already-tested code in `master`.

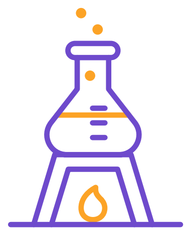


Start your GitLab free trial

Impact on collaboration

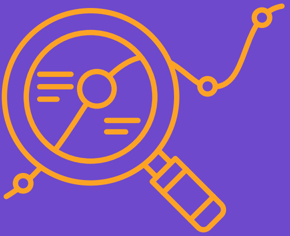
Coding in branches empowers team members to experiment and find innovative solutions to complex problems.

Teams can be creative without the fear of making the `master` branch unstable. Furthermore, team members can collaborate to ensure a solution works well before merging it to the `master` branch. Operations, quality, and security teams can review code before it gets deployed, ensuring that everyone has visibility and the opportunity to discuss ideas and propose any potential problems before shipping.



Part Five

Conduct regular code reviews



Overview

Implementing a culture of regular code reviews ensures continuous improvement and prevents unstable code. All team members should be welcome to review anyone's code and offer suggestions.

As soon as there is any code to review, a team member should assign the code review to an individual familiar with the project, a member of the same team, or to a domain expert.

When conducting a code review, team members should:

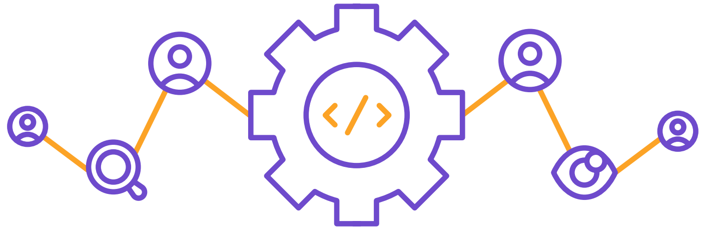
- ✓ **Explain** which changes are necessary (e.g. fixes a bug, improves the user experience, refactors existing code).
- ✓ **Prefix** comments with "Not blocking:" if there are small, non-mandatory improvements, helping the author understand that the suggestion is optional and can be resolved immediately or in another iteration.
- ✓ **Offer** alternative implementations, but assume the author already considered them (e.g. "What do you think about using a custom validator here?").
- ✓ Try to be thorough to reduce the number of iterations.
- ✓ **Identify** ways to simplify the code while still solving the problem.

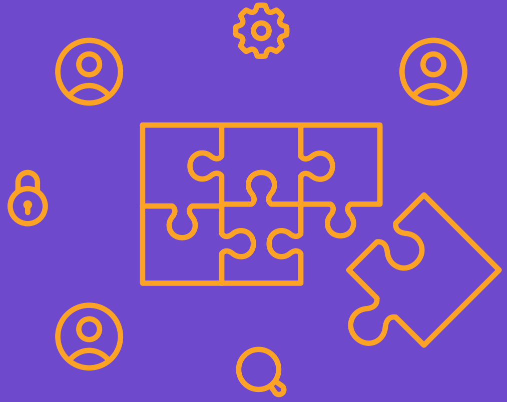
Start your GitLab free trial

Impact on collaboration

Code reviews act as a second opinion on a solution and implementation and as an extra pair of eyes looking for bugs, logic problems, or uncovered edge cases.

Code reviews help alleviate challenges when racing towards a release while carrying a burdensome issue. Reviewing solutions and offering suggestions can help team members review in tandem.





By collaborating in code reviews, teammates learn different coding practices, workflow techniques, and new ways of approaching problems, which increases innovation and efficiency and decreases knowledge silos.

Start your GitLab free trial

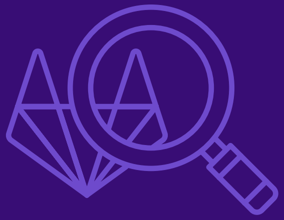
Version control and collaboration with GitLab

GitLab is a comprehensive version control and collaboration (VC&C) solution to deliver better software faster.

As a Git-based web repository that aggregates all development milestones and metadata, GitLab enables clear code reviews, asset version control, feedback loops, and powerful branching patterns to streamline software delivery. GitLab helps teams deliver faster and more efficiently with increased compliance.

GitLab version control and collaboration provides teams with increased **visibility, collaboration, and velocity.**

Seamless collaboration is the key to delivering better products faster, and GitLab's version control solution can help your team achieve business goals and accelerate delivery.



Visibility

GitLab Insights and Compliance Dashboard can aggregate and compose dashboards to capture bottlenecks, apply solutions, and foster continuous improvement.

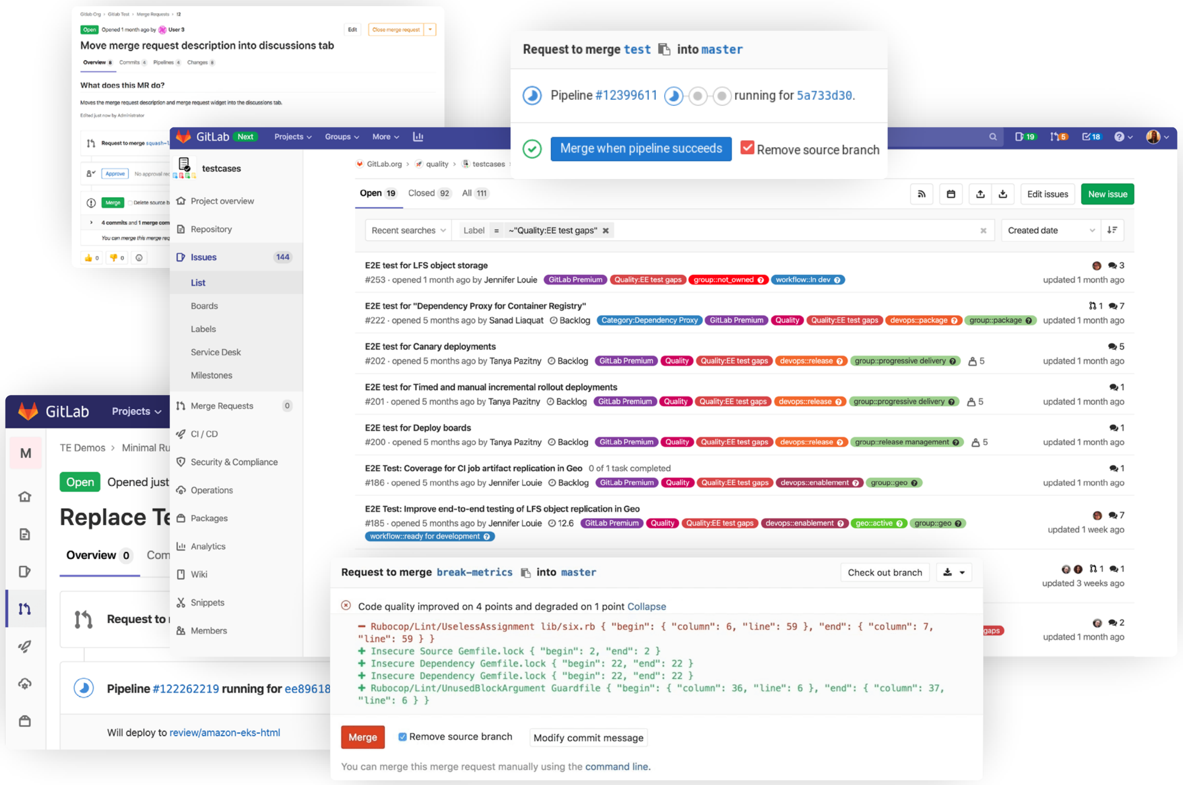
Collaboration

GitLab's Merge Requests and Code Quality provide the perfect canvas to manage all aspects of changing code and project assets, enabling reviews to provide feedback and immediate action to improve code.

Velocity

GitLab Merge Requests facilitate automated merge approvals and issue closing to automate tedious tasks, enabling developers to focus on shipping features.

Version control in GitLab helps your development team share, collaborate, and maximize productivity with unparalleled source code management. GitLab makes source code management easy by assembling all critical project files and assets in one interface, delivering it through a single application for the entire software development lifecycle.



Move merge request description into discussions tab

What does this MR do?

Request to merge test into master

Pipeline #12399611 running for 5a733d30.

Merge when pipeline succeeds ☑ Remove source branch

GitLab.org › quality › testcases

Open 19 Closed 92 All 111

Edit issues New issue

Recent searches Label = ~"Quality:EE test gaps" Created date

E2E test for LFS object storage
#253 · opened 1 month ago by Jennifer Louie GitLab Premium Quality:EE test gaps group::not_owned workflow::In dev — updated 1 month ago

E2E test for "Dependency Proxy for Container Registry"
#222 · opened 5 months ago by Sanad Liaquat Backlog Category:Dependency Proxy GitLab Premium Quality Quality:EE test gaps devops::package group::package — updated 1 month ago

E2E test for Canary deployments
#202 · opened 5 months ago by Tanya Pazitny Backlog GitLab Premium Quality Quality:EE test gaps devops::release group::progressive delivery — updated 1 month ago

E2E test for Timed and manual incremental rollout deployments
#201 · opened 5 months ago by Tanya Pazitny Backlog GitLab Premium Quality Quality:EE test gaps devops::release group::progressive delivery — updated 1 month ago

E2E test for Deploy boards
#200 · opened 5 months ago by Tanya Pazitny Backlog GitLab Premium Quality Quality:EE test gaps devops::release group::release management — updated 1 month ago

E2E Test: Coverage for CI job artifact replication in Geo 0 of 1 task completed
#186 · opened 5 months ago by Jennifer Louie Backlog GitLab Premium Quality Quality:EE test gaps devops::enablement group::geo — updated 1 month ago

E2E Test: Improve end-to-end testing of LFS object replication in Geo
#185 · opened 5 months ago by Jennifer Louie 12.6 GitLab Premium Quality Quality:EE test gaps devops::enablement geo::active group::geo workflow::ready for development — updated 1 week ago

Request to merge break-metrics into master

Code quality improved on 4 points and degraded on 1 point Collapse

```
− Rubocop/Lint/UselessAssignment lib/six.rb { "begin": { "column": 6, "line": 59 }, "end": { "column": 7,
"line": 59 } }
+ Insecure Source Gemfile.lock { "begin": 2, "end": 2 }
+ Insecure Dependency Gemfile.lock { "begin": 22, "end": 22 }
+ Insecure Dependency Gemfile.lock { "begin": 22, "end": 22 }
+ Rubocop/Lint/UnusedBlockArgument Guardfile { "begin": { "column": 36, "line": 6 }, "end": { "column": 37,
"line": 6 } }
```

Merge ☑ Remove source branch Modify commit message

You can merge this merge request manually using the command line.

TE Demos › Minimal Ru…

Open Opened just…

Replace Te…

Pipeline #122262219 running for ee89618

Will deploy to review/amazon-eks-html

Try GitLab free

All GitLab features — free for 30 days.
GitLab is more than just version control and collaboration. 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