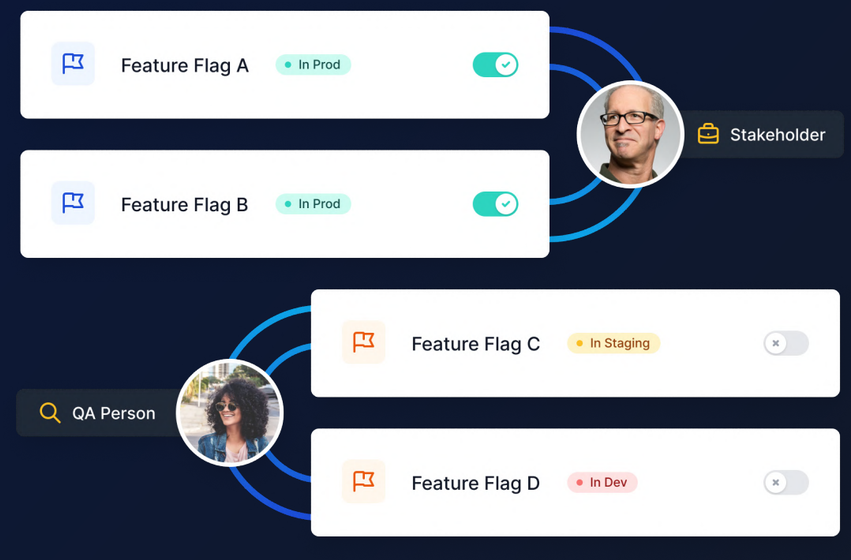
DevCycle

Feature Management For Modern Devs.



Transitioning To Trunk Based Development

Nothing makes an engineer’s life harder than dealing with dreaded merge hell. Having to rewrite code and spending hours trying to merge into main is tiresome, stressful, and wastes engineering resources.

Luckily, there’s a solution; Trunk-based development is a development strategy that only uses one branch. With trunk-based development, developers use short branches to make updates to the main trunk which is also known as a release branching strategy. The larger a team is, the shorter these branches should be since you want to have visibility of updates that are constantly merged to your main branch, also known as your trunk.

This helps with developers' workflow to increase productivity, minimizes the risk of bugs, and gives developers peace of mind when shipping new code. We’ll dive into when you can use trunk-based development, why you should use it, development alternatives, and how you can easily implement this strategy with the help of feature flags.

Table Of Contents

What Is Trunk Based Development?

Trunk-based development is a branching strategy where all developers work on the same "trunk" of shared code. The trunk is always in a releasable state, which means that at least once a day, developers must integrate their changes to the trunk. This is accomplished through short-lived feature branches related to project tasks.

This is a strategy that a software development team uses when writing, merging, and shipping code that eliminates merge hell. By keeping the number of lines changed small it is less likely for another developer's change to conflict. When working as a team, developers need to share their changes with each other. Branching is a strategy that helps teams manage their work. It defines how the team manages branches to get the most done without disrupting each other. The best branching strategy is trunk-based development.

In order to do trunk-based development, you need shorter branches, and feature flags naturally allow you to create short branches in a risk-free way. With feature flags, you can create small changes like adding part of a feature, fixing a bug, or adding a small drop-down to your site all in production. Your updates will be in main, but they just won't be visible. They also allow you to work on big changes. You can have larger features in main but have them hidden behind a feature flag. This allows you to slowly make updates to a feature in main, but have it disabled in production for your users since your trunk is always ready to go in production.

Feature Branches

A feature branch can last as long as needed, depending on the branching strategy, but for trunk-based development, these branches should last at most a day. The branch is often used by a single developer for their changes, but it can be shared with other developers as well. Feature branches are merged into the trunk when ready.

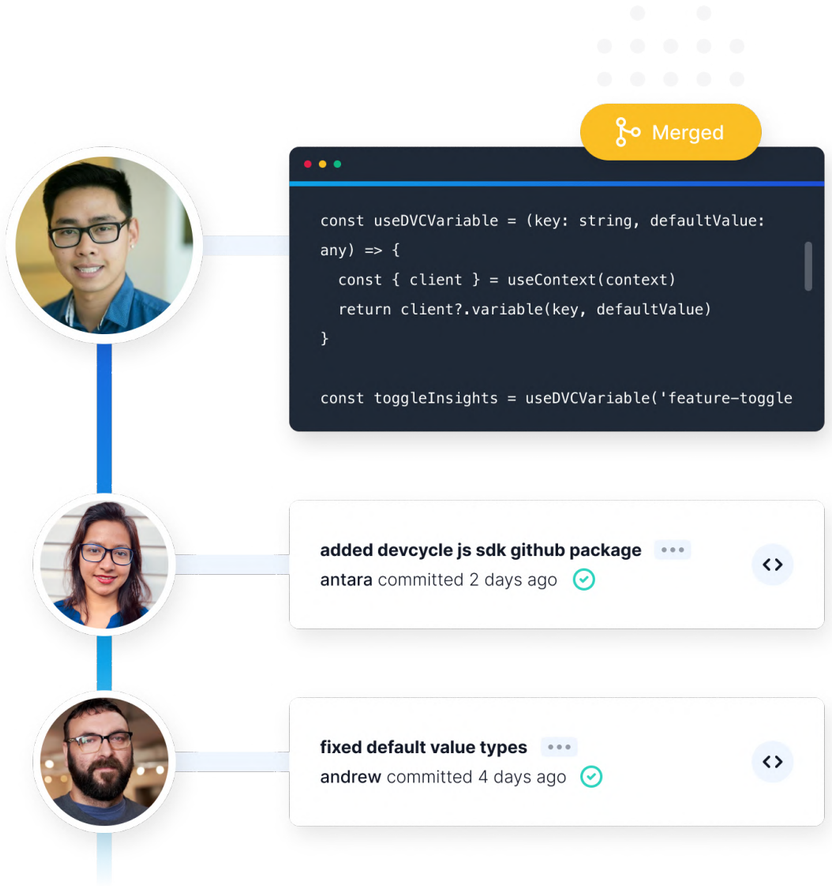
In order to use trunk-based development, you need a project to be broken down into tasks that take up less than a day's worth of effort. This means that each day a developer can merge their feature branch into the main trunk.

No matter what they do on their personal repo, at least once a day they must integrate their changes to the shared trunk. Developers are constantly checking in to see what the other members of the team are doing. This practice forces them to see changes and react to them. When this is constant, it forces collaboration around the quality and state of the codebase.

As continuous delivery and DevOps have become increasingly prominent and even necessary for many software development teams, Trunk-based has the tightest alignment with those modern delivery approaches. In fact, trunk-based development is a prerequisite for CI/CD.

Branching Strategy Use Cases

Accurate software delivery is a must, and the best way to ensure accuracy is to make sure everyone on your team is following the same process. A branching strategy is best for collaboration, efficiency, and meeting deadlines. Without a strategy or the wrong strategy, your team can lose hours or days of effort.



Daily Workflow

The typical day-to-day flow for developers includes changes that are small and not urgent; they usually make up the bulk of all the changes a developer will make. Your strategy for this flow must ensure proper coordination among developers and support all relevant policies. For example, if you have automated tests, pull requests, or deployment procedures, they should be coordinated in this workflow. All these changes are also managed externally in a workflow management system such as JIRA to allow for visibility across the team and to ensure everyone stays up to date with their tasks.

When working with JIRA, each ticket helps make up a small part of a larger task. Similarly, small feature flags can be created under a larger feature flag. This allows teams to work on code in main while having it hidden.

Focusing on completing one task a day not only helps with your development but increases productivity. People are more productive when they're focusing on one task. 20% of productivity is lost for every additional task someone tries to complete at once.

Emergencies

An emergency hotfix is an urgent bug fix that needs to be dealt with quickly. To ensure the developer can do this, the flow needs to account for their needs. This means that your process should have a way for the

developer to make an urgent change to production while still following your standard development workflow.

This strategy allows for the trunk to always be ready to go to production and therefore a hotfix and be merged in right away and released without the need for a regression test of all the other changes that are in the release but hidden by a feature flag.

Changes Of All Sizes

As the industry has evolved, the focus on small changes and limited batch sizes has increased. But there are times when you must make big, complex changes, and your branching strategy needs to accommodate these instances.

Experimental Changes

Developers want certainty about code changes, and if they're trying out a new library or framework, they may be concerned about how it integrates with their codebase. It's important to account for experimental changes when you're branching. For example, you may decide that your change won't be released immediately but still needs to be shared with other developers.

Types Of Branching Strategies

Git Branching Model

Every Git repository has a trunk, commonly called main today. There are many different branching strategies for developers to use when it comes to the trunk and the timing of changes landing on it. The timing of when code changes happen and the branching strategy used also depend on the exact version control system in use.

The Development Branch

The development branch is used to hold changes made by developers before they are ready to be deployed to a release branch. This branch is never removed. Commits to the development branch trigger deployments to the test environment. The development branch and trunk are integrated bi-directionally frequently so they’re constantly updated and integrated.

Release Branches

Release branches are the branches of the Git repository where the new feature is ready to be moved to production. A release branch can be either short-lived or long-lived depending on the strategy. In either case, the release branch reflects a set of changes that are being made to production.

During the development process, sometimes emergency bugs occur. When this happens, a hotfix branch can be created. It’s a short-lived branch that is used to fix bugs in the main product. Whether it be short-lived or long-lived, hotfix branches are more common in teams with explicitly versioned products.

Benefits Of Using Trunk Based Development

Developers can use trunk-based development to put continuous integration front and center in their workflow. It is the perfect way to work because you will not let your day go by without integrating your changes. This will make every developer think differently about how they approach their work, which then helps team members collaborate better. Continuous integration is also essential for innovating at a faster pace.

The key part of trunk-based development is the use of short-lived feature branches. These branches come directly from the trunk and are designed to be associated with a single piece of functionality that can be completed by a developer within a day.

Workflows around pull requests, automated build, and testing of branches are maintained. The difference is that the merged code goes directly into the trunk when approved. Modifying your work every day means taking small steps forward. You can't make sweeping changes to your code files and break your local compilation. Instead, make each step forward by making all of your changes in a few files. By making small continuous changes it minimizes the impact that an error in your code would cause in production.

Automated tests are run more often with short-lived branches, and end-to-end tests can catch small bugs before they become big. Sometimes you may want to make changes to a non-production environment first, make sure they work properly, and then move them to production. Other times, you may want a limited group of customers to see your changes first before going out to all of them. Trunk-based development with feature flags gives developers and product managers more control when releasing their changes. With feature flags, unfinished code can be deployed to production without end-users seeing it. Feature flags thus reduce the risk of deploying unfinished code to production while providing the opportunity to test code in production.

Pros And Cons Of Trunk And Branch Based Development

Trunk Based Development

Pros

One of the main benefits of trunk-based development is that your main branch is always in a release-ready state. This means you can release at any time without the stress of code not being ready. Since trunk-based development uses small yet frequent updates, these changes are easy to review and QA. This creates a high level of communication between developers as changes need to be reviewed daily.

Cons

With trunk-based development, bugs can creep in because full regression testing isn't done on each merge. In addition, developers need to wait for their small change to go through the automated build and test processes before merging.

Branch Based Development

Pros

With branch-based development, specific features can be developed in isolation from others and testing can be done on specific features in separate environments.

Cons

When merging back in the trunk, several problems can arise causing delays, new bugs, and requiring re-testing which takes up a lot of developer time and resources. In addition, reviewing pull requests can be a lengthy process due to the high quantity of code that has been added or changed.

Using Feature Flags For Trunk Based Development

In trunk-based development, you want changes to be visible in development for testing purposes, but not in production. When it comes to deploying in trunk-based development, you must make a distinction between deployment and release. The deployed code is moved to an environment for testing, while the released code is visible in the environment. The first step in linking an application setting to your branch is to create an abstraction seam. This technique will release your changes into a non-production environment without releasing them in the production environment. This gives you confidence that your changes won't be released in production unless you want them to be, but also allows you to release them in other branches.

Once you've used feature flags to manage releases between production and non-production environments, it's time to do an incremental release into production. You may do a canary release, which you start by releasing a change to internal or pilot users only. You observe the impact of this change and then release it to more and more customers until eventually it's released to your entire customer population.

Build vs. Buy

Engineering teams may think they could build their own version of a feature flag management system, but this process can take up a lot of time and resources. Building a strong feature flagging system takes focus away from the core competency of the organization and the team. However, it is still important for engineering teams to have access to this level of feature flag management. A software development team will want to find a feature flagging service as a way to make their life easier.

Using DevCycle Feature Flags

Trunk-based development is a powerful technique for developing better quality code. It reduces complexity by cutting down on the possibility of mistakes and streamlining collaboration between developers. DevCycle provides feature flags as a service to seamlessly release new code.

DevCycle enables you to precisely control the release of your features. You can set up a feature flag to be released on a specific date and time, which is ideal for planning a new launch. During this process, you have complete control over who can access the new feature. This release strategy also lets you test the latest versions of your software with a smaller, more targeted audience that is more likely to tolerate bugs or glitches as you work on the product.