## Branching off main

1. List all branches

Git branch

Or

Git branch -a

1. Create a new branch from main branch

Git branch my-branch

Git branch -a

Git log –oneline

1. Creating a branch based on another branch.(make sure you have the branch named new-feature)

Git branch new-feature-a new-feature

<https://www.boldare.com/blog/what-is-trunk-based-development/>

## Trunk based development

## Why is it worth getting familiar with trunk-based development?

Imagine you are working on an article with a few other colleagues. You all decide to split the work into sections and work on them individually. When the sections are all finished, you combine them into the final document. There is a good chance that you will have to spend a lot of time merging changes and the final document may still not live up to the standard you envisioned for it. Each section might be written using a different style, and duplicating ideas. The feedback on the whole document will likely be delivered very late and in the end some parts might have to be rewritten completely.

The alternative approach is to deliver the document by writing small sections that will be combined relatively fast with the rest of the article. In this approach, you focus on delivering smaller changes incrementally to get early feedback. More frequent integration could reduce duplication and improve the communication between collaborators. This approach could produce articles of higher quality much faster.

**While writing articles and code is vastly different, the chosen workflow method has a big impact on the final result**. **That’s why you should look into why trunk-based development may be important for your team.**

## What is trunk-based development? A definition

**Trunk-based development (TBD) is a source control workflow model that enables continuous integration.** Its key ideas are correlated with DevOps practice. The main goal of trunk-based development is to avoid long-lived branches by merging partial changes to the whole feature. To do so developers can commit directly to the main branch or use short-lived branches with an efficient code review process. By definition, branches should only last a couple of days ([source](https://trunkbaseddevelopment.com/)).

Merging partial changes **results in initial feedback, reduces merging complexity and reduces duplicated work** if other developers are working on related changes. When the merging process is efficient, the developer doesn’t lose focus by context switching to other tasks. Each change doesn’t break the build, because the main branch, often called the trunk, should be ready to deploy at any moment. There are practices, such as **automated testing, feature flags or branching** **by abstraction** that enable implementation of this workflow in the team.

## Benefits of using trunk-based development

### Reduced complexity

**One of the main trunk-based development benefits is the reduced complexity of merging different branches into one**. This approach aims to avoid merge hell, a situation when different pieces need to be combined for the first time which leads to unexpected bugs, integration issues and blocks the team from deploying. This could happen just before the end of the sprint, which also increases the level of stress. The more branches you have, the more complexity developers have to deal with. Using trunk-based development means that most of the developers’ time is spent on producing code, rather than performing merges.

### Increasing speed of delivery

Implementing and using trunk-based development in the long term could increase team discipline and a feeling of teamwork by establishing clear processes and giving more opportunities for collaboration. When properly implemented it can also increase the speed and predictability of delivery. According to the State of Devops 2021 report ([source](https://services.google.com/fh/files/misc/state-of-devops-2021.pdf)) produced by Google, **high performing organizations are more likely to have implemented trunk-based development**.

### Shortening the feedback loop

Building in short feedback cycles could also help verify initial design assumptions. Because the trunk should be always stable, the code is potentially releasable to customers for getting early feedback.

## Suggested practices for implementing TBD

**Before implementing trunk-based development some steps should be taken.** Technical requirements such as automation and a stable test suite are advised to keep the trunk green.

### Automation

**Automating the building, testing and deployment software are key enablers of trunk-based development.** It allows for making quick iterations while reducing the chance of breaking the main branch. The test suite should be fast so as not to block the iteration cycles. Automating the deployment process makes it more resilient, because manual deployment processes are error prone.

### Feature flags

Using feature flags you can decouple deploying the new code to the production environment from releasing the feature to the users. Feature flags (also called feature toggles) could be defined during the application building stage or controlled in run-time by external management software. Feature flags don’t come without their issues though. They increase the testing surface by adding new combinations of behavior. Reusing or not deleting unused feature flags could also lead to an increase in [technical debt](https://www.boldare.com/blog/how-much-technical-debt-is-acceptable/).

Using feature flags, you can decouple deploying the new code to the production environment from releasing the feature to users. Deploy is a preparation step, while release is a publication step. Canary release technique can be used to test new features on limited user groups.

### Branching by abstraction

Branching by abstraction is a technique of safely introducing large changes that won’t break the system. It’s done by introducing an abstraction between the existing implementation and the module that needs to be replaced. When the abstraction is in use, it can gradually switch to the new module. When the transition is completed, the old module is deleted.

This technique allows developers to split their work into smaller commits instead of creating a single merge request that could be difficult to manage. Branching by abstraction prevents introducing breaking changes and allows you to pause and resume the change over time, because the application is working throughout the process.

### Efficient code reviews

Good communication is a crucial element of high-performing teams and it’s very important while trying to implement a trunk-based development approach. There are techniques such as pair programming or mob programming that encourage more communication between developers in the team. Those techniques allow developers to contribute and familiarize themselves with the changes without an extensive and time-consuming code review process.

Working with pull request

Finding differences

Git diff

<https://www.cloudbees.com/blog/git-diff-a-complete-comparison-tutorial-for-git>

FAQ:what is the difference between fork and clone

**A fork creates a completely independent copy of Git repository.** **In contrast to a fork, a Git clone creates a linked copy that will continue to synchronize with the target repository**.

## Feature flag

<https://www.atlassian.com/continuous-delivery/principles/feature-flags>

## Feature flag use cases

The novel utility of feature flags enables a variety of creative uses cases for industrious teams. The following examples highlight some popular applications of feature flags in an agile environment.

### Product testing

Feature flags can be used to gradually release new product features. It can be unclear upfront if a proposed new feature will be adopted by users and is worth the return on investment. Teams can release a new product feature or a partial idea of the feature in a feature flag and deploy it to a subset of users to gather feedback. The subset of users may be vocal power users who are happy to beta test and review. If the new product idea proves to be a hit, the development team can then roll out the feature flag to a larger user base. If instead the new product idea turns out to be a flop, the development team can easily disable the feature flag and later remove it from the codebase.

### Conducting experiments

Experiments or A/B testing are a primary feature flag example. In their simplest form, feature flags act as a toggle of “on” and “off” state for a feature. Advanced feature flags utilize multiple flags at once to activate different experiences for subsets of users. For example, imagine you divide your user base into thirds. Each third receives its unique flag and user experience. You can then measure the performance of these three flags against each other to determine the final committed version.

### Migrations

There are times when an application needs a data migration that requires dependent application code changes. These scenarios are sensitive, multi-phase deployment tasks. A database field may be modified, removed, or added in an application database. If the application code is not prepared for this database change, it causes failures and errors. If this happens, it requires a coordinated deploy between the database changes and the application code.

Feature flags help to ease the complexity of this scenario by allowing teams to prepare in advance application changes in a feature flag. Once the team makes the changes to the database, they can immediately switch toggle the feature flag to match the application code. This removes the risk and delay of waiting to deploy the new application code and possibly having the deployment fail and desync the application from the database.

### Canary launches

A canary in this context refers to an old, morbid practice where coal miners brought canary birds into the coal mine to detect carbon monoxide. The birds have higher metabolism rates and rapid breathing rates, which led them to succumb to carbon monoxide before the miners.

Canary launches in software development occur when a new feature or code change is deployed to a small subset of users to monitor its behavior before releasing it to the full set of users. If the new feature shows any indication of errors or failure, it is automatically rolled back. Feature flags are essential to this process since they restrict the audience pool and can toggle off features easily.

### System outage

A feature flag can also be used as a system outage tool. A web application may utilize a feature flag to “switch off” the entire website for maintenance or downtime. The feature flag can be instrumented throughout the codebase to push sensitive transactions and display outage content to the end users. This can be incredibly helpful when doing sensitive deployments or if an unexpected issue is found and needs to be urgently resolved. It gives teams the confidence and capability to take a controlled outage if deemed necessary.

### Continuous deployment

Feature flags can be used as an integral component to build a truly continuous deployment system. Continuous deployment is an automated pipeline that takes new code from developers and automatically deploys it to production end users. Continuous deployment depends on layers of automated tests that verify new code behaves as expected against a matching specification as it moves through the pipeline.

Feature flags make it safer to deploy continuously by separating code changes from revealing features to users. New code can automatically merge and deploy to production and then wait behind a feature flag. The continuous deployment system can monitor user behavior and traffic, then automatically activate the feature flag. Inversely, the continuous deployment system can monitor the new feature flag code to see if it behaves as expected, and roll it back if not.

## FAQ:Feature branches vs. feature flags

Though they are similarly named, feature branches and feature flags are quite different. [Feature branches are a workflow pattern](https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow) that is relevant to Git source control repositories. When developers begin a new product feature, they create a new corresponding Git branch that deviates from the main branch and will contain all the code for the new feature. When developers finish a new feature, they merge the feature branch back into the main and deploy. Feature branches are distinct and standalone from feature flags.

Feature flags can be used with feature branches. To create a feature flag, a developer creates a feature branch and commits the new feature flag code to the feature branch. Once instrumented, the feature branch is merged and deployed and the feature flag then becomes accessible on production.

During feature branch development a project can become long-lived, or deviate from the main branch and contain numerous commits. Developers need to ensure a feature branch is continuously updated so when it’s ready to merge there are minimal conflicts. It takes developers extra effort to ensure a feature branch is continuously updated so that once it’s time to merge the feature branch, there are minimal conflicts. Feature flags can be used to remedy this scenario. Instead of creating a long-lived feature branch and doing a large merge when the new feature is ready, a feature flag can be created and immediately merged to main

FAQ: What is the difference between deploy and release

Deploy is the preparation , release is for production

FAQ: What is the difference between dev and test environment?

Development: The program or component is developed on a development system. This development environment might have no testing capabilities. Testing: Once the software developer thinks it is ready, the product is copied to a test environment, to verify it works as expected.